Semantik I

Proseminar SS 2004
CIS, Universität München

Hans Leiß

22. Oktober 2004

Zusammenfassung

Es werden exemplarische Grammatiken zu Nominalphrasen, einfachen Sätzen, verallgemeinerten Quantoren im Singular und Plural, Temporal- und Lokaladverbien angegeben sowie passende Interpretationen und Auswertungsfunktionen dafür definiert. Es wird gezeigt, wie man mit einem Parsergenerator daraus automatisch Programme zur Analyse und Auswertung korrekter Ausdrücke erzeugen kann.

Die Teile des Proseminars, die die Semantik nach Montague vorstellten, sind hier nicht dokumentiert.

Inhaltsverzeichnis

1 Einleitung

In der Semantik geht es um die Bedeutung von sprachlichen Ausdrücken. Die Bedeutung eines sprachlichen Ausdrucks hängt von der Form oder dem Aufbau des Ausdrucks ab, aber auch von der Verwendungssituation.

Sprachliche Ausdrücke sind besondere Zeichen, mit denen die Menschen sich untereinander verständigen; daher werden sie normalerweise verwendet, um bestimmte Verständigungziele zu erreichen. Solche Ziele sind etwa:

(a) anderen eigene Gedanken und Gefühle mitzuteilen, die für diese nicht unmittelbar wahrnehmbar sind,

(b) andere auf wahrnehmbare Erscheinungen aufmerksam zu machen,

(c) andere zu sprachlichen oder Verhaltensäußerungen zu bewegen.

Die Verwendung von Zeichen für solche und ähnliche Zwecke setzt voraus, daß diese Zeichen beiden Beteiligten, dem „Sprecher", der die Äußerung macht, und dem „Hörer", der sie –vielleicht an einem anderen Ort und zu einer anderen Zeit– wahrnimmt, etwas bedeuten und von dem sie zumindest glauben, daß es für beide „im Wesentlichen dasselbe" ist. Dieses unterstellte Gemeinsame der sprachlichen (wie der anderen) Zeichen läßt sich nicht einfach willkürlich festlegen oder ändern, sondern wird durch einen komplizierten kulturellen Prozeß im Laufe der Zeit weiterentwickelt und an neue Generationen weitergeleitet.

Es ist naheliegend, daß hiervon nur ein kleiner Teil im Rahmen einer Theorie erfaßt werden kann. In der Semantik schliest man sowohl die kulturellen Hintergründe als auch die konkreten Verständigungsziele einer Äußerungssituation weitgehend aus. Man unterstellt stattdessen, daß die Beziehung zwischen der Form der Ausdrücke und dem, was diesen Ausdrücken in der Vorstellung der Beteiligten entspicht, in sinnvoller Weise isoliert betrachtet werden kann.

Damit bleibt als eine wesentliche Aufgabe der Semantik, die Beziehung zwischen komplexen Ausdrücken und ihren Bedeutungen auf angenommene Beziehungen zwischen elementaren Ausdrücken und deren Bedeutungen zurückzuführen, sowie zu bestimmen, wie die Bedeutung sprachlicher Ausdrücke von angenommenen Eigenschaften der Äußerungssituation abhängt.

Von den Äußerungssituationen kann in gewissem Grade abgesehen werden, weil sich für bestimmte Verständigungsziele bestimmte Ausdrucksformen –wie Bezeichnungen, Behauptungen, Fragen, Befehle– herausgebildet haben, an denen man die Ziele normalerweise ablesen kann.

2 Kategorien von Ausdrücken und Bedeutungen

Es ist m.E. lehrreich, sich den Beginn der theoretischen Untersuchungen zu den natürlichen Sprachen anzusehen, insbesondere die des Aristoteles. Man sieht daran, daß

(a) gleich zu Beginn eine ganze Reihe unterschiedlicher sprachlicher Ausdrucksformen unterschieden wurde,

(b) viele dieser Unterscheidungen im Verlauf der historischen Entwicklung in den Hintergrund gerückt wurden, weil sie für die formalen Sprachen der Logik und Programmierung von geringerer Bedeutung zu sein schienen,

(c) manche der unterschiedenen Ausdrucksarten eine wesentliche Rolle bei der Organisation der menschlichen Auffassung von der Alltagswelt spielen,

(d) das theoretische Verständnis der Funktionsweise menschlicher Sprache noch keineswegs vollständig ist, und

(e) die Schwierigkeiten, die Bedeutungsaspekte menschlicher Sprachen einer maschinellen Behandlung zugänglich zu machen, erheblich sind.

2.1 Die Kategorien von Aristoteles

Aristoteles unterscheidet zwei wesentliche Funktionen von Ausdrücken: das *Benennen* und das *Aussagen*.

Ein *Nomen* (im Nominativ des Singulars), ὄνομα, kann etwas Einzelnes benennen, z.B. „Kallippos“ einen bestimmten Menschen, etwas Allgemeines, z.B. „Mensch“ die verschiedenen einzelnen Menschen, oder etwas Abstraktes, z.B. „Gesundheit“ den Zustand, gesund zu sein. Ein *Verb* (im Präsens der dritten Person des Singulars), ῥῆμα, benennt das von etwas anderem Gesagte und sein Vorliegen bei der Äußerung, z.B. „leidet“. [1]

Durch die Verbindung eines Nomens mit einem Verb entsteht eine *einfache Aussage*, ein λόγος ἀποφαντικός, die *wahr* oder *falsch* sein kann. Aus einfachen Aussagen lassen sich durch Verbindung mit z.B. „nicht”, οὐ, oder „und“, καί, *zusammengesetzte Aussagen* bilden.

Wir betrachten nun etwas näher die Darstellung der Sprache, die Aristoteles in seiner Schrift über die „Kategorien“ gibt. Er beginnt mit der Unterscheidung zweier Arten von Grundaussagen (Kap.2, 1a 20):

(a) τινὸς καθ' ὑποκειμένου τινὸς λέγεται, „x wird über y ausgesagt“, kurz: $x(y)$,

(b) τινὸς ἐν ὑποκειμένου τινὸς ἐστίν, „x kommt an y vor“, kurz: $x[y]$

Die Grundaussagen $x(y)$ sind die, bei denen das Verb x über das „Zugrundeliegende“, τὸ ὑποκείμενον, oder das „Subjekt“ y ausgesagt wird. Das Verb x kann hierbei ein Vollverb sein, wie in „ein Mensch läuft“, ἄνθρωπος τρέχει, oder aus

[1] Andere Formen der Nomen und Verben gelten ausdrücklich nicht als ὄνομα und ῥῆμα (Hermeneutik, Kap.2 und 3).

der Verbindung eines Nomens mit der (oft nicht explizit vorhandenen) Kopula „ist“, ἐστίν, entstehen, wie in „ein Mensch ist ein Lebewesen“, ἄνθρωπος ζῶον (ἐστίν). Die Grundaussagen $x[y]$ sind wohl die Prädikativsätze, wo das Prädikat x aus der Verbindung eines Adjektivs mit der (wieder oft weggelassenen) Kopula entsteht, z.B. „der Körper ist weiß“, σωμα λευκός (ἐστίν).

Aristoteles unterscheidet dann vier Arten von „Seiendem“ nach den Möglichkeiten, wie es als x oder y in den Aussagen $x(y)$ und $x[y]$ verwendet werden kann:

(a) Allgemeinbegriffe A: das, was von etwas ausgesagt wird, aber an nichts vorkommt. Falls $x : A$ (d.h. x ein Allgemeinbegriff ist, und y ein Seiendes), so ist $x(y)$ eine Aussage, aber $x[y]$ ist keine Aussage.

(b) Eigenschaften oder Qualitäten E: das, was an etwas vorkommt, aber von nichts ausgesagt wird. Falls $x : E$, so ist $x(y)$ keine Aussage, aber $x[y]$ ist eine Aussage.

(c) Fähigkeiten F: das, was an etwas vorkommt und von etwas ausgesagt wird: Falls $x : F$, so sind $x(y)$ und $x[y]$ Aussagen.

(d) Objekte (bzw. Subjekte) I: das, was nicht an etwas vorkommt und auch nicht von etwas ausgesagt wird. Falls $x : I$, so sind weder $x(y)$ noch $x[y]$ Aussagen.

Die Klassifikation benutzt nur das erste Argument x der Aussageformen. Grammatische und ontologische Begriffe werden aber nicht klar getrennt.

In Kap.4 unterscheidet er *Aussagen* oder *das, was in einer Verbindung eine Bedeutung hat* von *Ausdrücken*, die auch unverbunden eine Bedeutung haben, aber erst durch Verbindung eine Aussage bilden (Kap.2, 1a,15; Kap.4, 2a,5).

Dann stellt er eine Liste von *Kategorien* der bedeutungstragenden Ausdrücke auf, mit denen zusammengesetzte Aussagen gebildet werden können:

Nr.	Name		gramm. Begriff	Beispiel
1.	οὐσία	seiendes Wesen	Gemeinnamen	Mensch
2.	ποσὸν	wieviel	Maßausdruck	zwei Ellen
3.	ποιὸν	wie beschaffen	Adjektiv	weiß
4.	πρός τι	im Verhältnis zu	Vergleichsausdrücke	doppelt, größer als
5.	ποὺ	wo	Ortsadverbiale	auf dem Markt
6.	ποτὲ	wann	Zeitadverbiale	gestern
7.	κεῖσθαι	sich (ver)halten	Intrans. Verben (medium)	liegt, sitzt da
8.	ἔχειν	haben	Zustandsverben	hat Schuhe an
9.	ποιεῖν	tun	Tätigkeitsverben (aktiv)	schneiden
10.	πάσχειν	leiden	Verben (passiv)	geschnitten werden

Die Kategorien enthalten Ausdrücke, die weder wahr noch falsch sein können: sie haben eine Bedeutung, die kein Wahrheitswert ist; es sind also keine Sätze.

Spätere Bemerkungen deuten an, daß die οὐσία auch die Individuen umfassen und eher einfach die *Subjekte* von Aussagen sind.

Nach Kap.1 hat jedes Wort (ὄνομα) mindestens eine Erklärung oder *Definition* (λόγος). Daneben werden vom Wort verschiedene Formen *abgeleitet*. Kap.5 führt zwei (metasprachliche) Sprechweisen ein, um das zu berücksichtigen:

(a) τὸν ὄνομα κατηγορεῖσθαι τοῦ ὑποκειμένου: den Namen von etwas (aus)sagen,

(b) τὸν λόγον κατηγορεῖσται τοῦ ὑποκειμένου: den Begriff von etwas aussagen.

Mit „den Namen aussagen" ist anscheinend nicht das *Nennen* gemeint, sondern eher, daß das Wort selber als Bezeichnung dient, nicht eine abgeleitete Form. Für die Grundaussagen $x(y)$ und $x[y]$ soll dann gelten (Kap.5 2a,20 und 2a,27):

- Falls „x von y ausgesagt wird", also $x(y)$, so wird (a) der Name x von y ausgesagt: „*y ist ein x*" ist grammatisch korrekt, und (b) der Begriff x von y ausgesagt: es wird behauptet, daß y die definierende Eigenschaft $Def(x)$ von x erfüllt.

  Beispiel: In „der Mensch ist ein Tier", ζῶον(ἄνθρωπος), wird (a) das Nomen im Nominativ des Singulars verwendet und (b) behauptet, daß die Definition von „Lebewesen" auf Menschen zutrifft.

  Das könnte man vielleicht auch so interpretieren: Aussagen $x(y)$ kann man nur in Kontexten machen, die eine Erklärung $Def(x) := \lambda y.\varphi(y)$ festlegen; andernfalls –bei „homonymem" x, wie ζῶον = (1. Lebewesen, 2. Bild)–, ist nicht klar, was behauptet wird. Dann würde Aristoteles hier die Konversionsregel des λ-Kalküls vorwegnehmen:

  $$(\beta)\ \frac{\Gamma, x = \lambda y.\varphi \vdash x(y)}{\Gamma, x = \lambda y.\varphi \vdash \varphi}$$

- Falls „x an y vorkomt", also $x[y]$ ausgesagt wird, so wird (a) normalerweise der Name x nicht von y ausgesagt, manchmal aber doch, und (b) niemals behauptet, daß y die definierende Eigenschaft $Def(x)$ erfüllt.

  Beispiel: das Weiß (τὸ λευκὸν) kommt an einem Körper (σώμα) vor, also λευκὸν[σώμα], und hier „wird der Name ausgesagt", d.h. σώμα λευκόν ist grammatisch korrekt; es wird aber nicht behauptet, daß der Körper ein Weiß oder eine Farbe sei (die Definition erfüllt).

  Im Normalfall wird „nicht der Name ausgesagt": die Schriftkunde kommt am Schriftkundigen vor, ἡ γραμματικῆ[ὁ γραμματικὸς], aber γραμματικὸς γραμματικῆ ist ungrammatisch, man braucht eine abgeleitete Wortform, ein Adjektiv. (Oder: der Schriftkundige *hat* Schriftkunde, aber *ist* keine Schriftkunde.)

Aristoteles beachtet also die Mehrdeutigkeit von Wörtern und Ableitungen zwischen Wörtern, die zu anderen Bedeutungen führen können, sagt aber nicht, was für ihn die Grundformen und was die abgeleiteten Formen sind.

Bemerkung 2.1 Das Adjektiv „weiß“ ist hier also kein Name des Bereichs D_A aller weißen Objekte, oder zumindest ist die Komprehension $\{y \mid \text{weiß}[y]\}$ nicht die Farbe Weiß. Das ist bei $\{y \mid x(y)\}$ insofern anders, als die Kategorien disjunkte Bereiche haben; dann kann man sie mit ihrem Umfang identifizieren. Aber wenn man neue Typen wie $\sigma \times \tau = \{\langle a, b\rangle \mid a : \sigma \wedge b : \tau\}$ bildet, kommt es doch sehr darauf an, welche definierenden Bedingungen erlaubt sein sollen.

2.1.1 Wesen: Objekte, Arten und Gattungen

Antworten auf die Frage „Was ist das?“ (τὶ ἔστι), oder „Was ist sein Wesen?“ sind *Gemeinnamen*, die (Arten von) *Objekte*(n) oder *Wesenheiten*, z.B. Lebe„wesen“, bezeichnen.

Unter den Objekten (ὀυσίαι) wird zwischen Objekten erster Stufe und Objekten zweiter Stufe unterschieden. *Erststufige Objekte* (πρῶται ὀυσία) sind diejenigen, die weder in der Position des Prädikates x von Aussagen der Form $x(y)$, noch in der von Aussagen der Form $x[y]$ vorkommen[2]. *Zweitstufige Objekte* (δεύτεραι ὀυσίαι) sind die Arten (εἴδη) und Gattungen (γένη) der erstufigen Objekte[3]. Die zweitstufigen Objekte klassifizieren die erststufigen Objekte[4].

Auf dem Bereich der zweitstufigen Objekte ist eine endliche[5] partielle Anordnung $\leq$ gegeben (Kap.3, 1b16). Die Ausagen der Form $x(y)$ sind bezüglich dieser Anordnung monoton im Prädikat: aus $x(y)$ und $x \leq x'$ folgt $x'(y)$.

Die Gattungen werden von ihren Arten im Sinne von $x(y)$ ausgesagt. Unterscheiden wir zwischen dem zweitstufigen Objekt x und dem Bereich (Domain) D_x seiner Elemente (d.h. zwischen dem Begriff und seinem Umfang), so umfaßt die Prädikation $x(y)$ also zwei Lesarten der Kopula „ist ein“, nämlich

$$x(y) \quad = \quad \begin{cases} y \in D_x, & \text{falls } y \text{ erststufig ist,} \\ D_y \subseteq D_x, & \text{falls } y \text{ zweitstufig ist.} \end{cases}$$

Die Arten einer Gattung werden durch *definierende* (oder *wesentliche*) *Eigenschaften* bestimmt und bilden eine disjunkte Zerlegung der Gattung.[6] Die artbildenden *Unterschiede* (διαφοραί) sind gattungsspezifisch[7] und auf die untergeordneten Gattungen übertragbar[8]. Die Unterschiede sind Prädikate x im Sinne

[2]Kap.5, 2a11: Οὐσία δέ ἐστιν ἡ κυριώτατά τε καὶ πρώτως καί μάλιστα λεγομένη, ἣ μήτε καθ′ ὑποκειμένου τινὸς λέγεται μήτε ἐν ὑποκειμένω τινί ἐστιν.

[3]Kap.5, 2a14: δεύτεραι δὲ οὐσίαι λέγονταται, ἐν οἷς εἴδεσιν αἱ πρώτως οὐσίαι λεγόμεναι ὑπάρχουσιν, ταῦτά τε καὶ τὰ τῶν εἰδῶν τούτων γένη.

[4]Kap.5, 2b30: μόνα γὰρ δηλοῖ τὴν πρῶτην οὐσίαν τῶν κατηγορουμένων.

[5]Immerhin soll ja jedes Objekt ein *genus proximum*, eine bezüglich $\leq$ nächste Gattung haben. Schon Porphyrios sagt, es gebe nur eine begrenzte Zahl von Gattungen. Aristoteles sagt es selbst, glaube ich, auch irgendwo in der Metaphysik. Andererseits sind die „zweifüßigen Lebewesen“ eine Art der Lebewesen, vielleicht auch für jedes n die „n-füßigen Lebewesen“.

[6]Daneben gibt es zufällige Eigenschaften, Attribute, die Unterschiede innerhalb einer Art ausdrücken und verschiedene Arten schneiden können.

[7]Kap.3, 1b16: τῶν ἑτερογενῶν καὶ μὴ ὑπ′ ἄλληλα τεταγμένων ἕτεραι τῶ εἴδει καὶ αἱ διαφοραίυ.

[8]Kap.3, 1b20: τῶν δέ γε ὑπ ἄλληλα γενῶν οὐδὲν κωλύει τὰς αὐτὰς διαφορὰς ε ἶναι.

der Aussagen $x(y)$, sie kommen nicht in Aussagen $x[y]$ vor; es wird stets ihr Begriff mit ausgesagt.[9]

Alle Wesen, erst- und zweitstufige, treten *nicht* „an“ etwas Zugrundeliegendem auf[10]: falls $x(z)$ oder $z(x)$, so ist $x[y]$ ausgeschlossen. (Die Attributwerte x sind keine Prädikate oder Objekte.)[11] Insbesondere treffen Gattungen und Unterschiede –anders als die Attributwerte– nicht in höhrem oder geringerem Grad auf ein Individuum zu, sondern entweder treffen sie zu oder nicht, d.h. sie sind zweiwertige Prädikate (Kap.5, 3b32).

Die erststufigen Objekte haben die Besonderheit, „Gegensätzliches anzunehmen“, d.h. eine Eigenschaft zu verschiedenen Zeiten zu haben bzw. nicht zu haben. Derselbe Gegenstand ist mal weiß und mal schwarz, mal warm und mal kalt, mal krank und mal gesund (Kap.5, 4a10f). Sie behalten ihre Identität (sowie ihr Wesen und die definierenden Artunterschiede) über die Zeit hinweg, aber ihre sonstigen Qualitäten wechseln sie mit der Zeit. Außerdem sind die erststufigen Objekte unteilbar und „eins der Anzahl nach“, während zweitstufige es nicht sind: zwar kann „Mensch“ auch zum Benennen verwendet werden, trifft aber auf vieles Einzelne zu.

In moderner Terminologie könnte man sagen: Aristoteles hat ein endliches System von Typen σ (den Gattungen), das durch eine Subtyprelation $\leq$ partiell geordnet ist, und nimmt getypte Unterscheidungsprädikate an, die sich auf Subtypen vererben: falls $U : \sigma \to \mathsf{bool}$ und $\tau \leq \sigma$, so ist auch $U : \tau \to \mathsf{bool}$.

Unter den einfachen Aussagen $x(y)$ spielen die Typaussagen $y : \sigma$ (statt $\sigma(y)$) und die Unterscheidungen $U(y)$ eine besondere Rolle, da ihr Wahrheitswert von der Zeit unabhängig ist. Bei anderen Aussagen, den Aussagen $x(y)$ mit Vollverb x und den Attributangaben $x[y]$, hängt er von der Zeit der Äußerung ab.

Eine Art τ wird mit durch eine Konjunktion von Aussagen $U(y)$ und $\overline{U'}(y)$ aus einem Typ σ ausgesondert:

$$y : \tau \iff y : \sigma \wedge \bigwedge_{i \in I} U_i(y) \wedge \bigwedge_{j \in J} \overline{U_j}(y).$$

Diese Unterschiede (oder ihr Komplement) umfassen die jeweilige Art[12]; die in dieser „Definition“ nicht vorkommenden Unterschiede vererben sich dann wie oben erwähnt von der Gattung auf die Art.

[9] καὶ ἣ διαφορὰ τῶν μὴ ἐν ὑποκειμένιω ἐστίν· τὸ γὰρ πεζὸν καὶ τὸ δίπουν καθ′ ὑποκειμένου μὲν λέγεται τοῦ ἀνθρώπω, ἐν ὑποκειμένω δὲ οὐκ ἔστιν

[10] Kap. 5,3a7 κοινὸν δὲ κατὰ πάσης οὐσίας τὸ μὴ ἐν ὑποκειμένω ε῏ιναι.

[11] Andererseits wird von den zweitstufigen Objekten gesagt, daß sie keine Individuen sind, sondern „eher ein *wie*-beschaffen“ anzeigen (Kap 5, 3b15): „Art und Gattung sagen über das Wesen das wie-beschaffen aus.“: τὸ δὲ ε῏ιδος καὶ τὸ γένος περὶ οὐσίαν τὸ ποιὸν ἀφορίζει, ποιὰν γάρ τινα οὐσίαν σημαίνει. (Kap.5, 3b20) In gewissem Sinne sind also Art und Gattung doch Werte x im Sinne von $x[y]$ bezüglich des Attributs „Wesen“. (Die Analogie ist sozusagen: Welche *Farbe* hat dies? – *weiß*. Welches *Wesen* hat dies? – *das des Menschen*.)

[12] So kann man jedenfalls Kap.5, 3b1, verstehen: ὡσαύτως δὲ καὶ αἱ διαφοραὶ καὶ κατὰ τῶν εἰδῶν καὶ κατὰ τῶν ἀτόμων κατηγοροῦνται.

2.1.2 Quantitäten

Antworten auf die Frage „Wieviel ist das?“ (ποσὸν) sind *Maßausdrücke*, die *Quantitäten*, *Größen* bezeichen. Aristoteles unterscheidet Quantitäten nach ihren Teilbarkeitseigenschaften in zwei Gruppen:

(a) *diskrete Größen* (oder Abgegrenztes, διωρισμὲνον), deren Teile sich nicht an einer gemeinsamen Grenze (ὅρος) berühren; hierzu zählt er *Zahl* und *gesprochene Rede* (mit Summanden bzw. Silben als Teilen), und

(b) *kontinuierliche Größen* oder Zusammenhängendes (συνεχὲς), deren Teile sich an einer Grenze berühren (συνάπτειν); hierzu zählt er: *Linie* (γράμμα), *Fläche* (ἐπιφάνεια), *Körper* (σωμα), *Zeit* (χρόνος) und *Raum* (τόπος), bei deren Teilung man Punkt, Gerade, Fläche, Jetzt, Fläche als Grenze erhält.

Bei manchen Größen haben die Teile eine *Reihenfolge* (τάξις), z.B. bei Zahl und Zeit, bei anderen eine *relative Lage zueinander* (ϑέσις), z.B. bei Linie, Fläche.

Die sprachlichen Ausdrücke, die Größen als Werte haben, scheinen Maßausdrücke der Form „Zahlausdruck mal Maßeinheit“ zu sein; jedenfalls ist „zwei Ellen“ (δίπηχυ) das einzige konkrete Beispiel (5b 13, 5b 25, 6a 19).

Als charakteristische Besonderheit der Größen nennt Aristoteles (6a 25), daß man sie mit „gleich und ungleich“ (ἴσον καὶ ἄνισον) vergleicht, während man z.B. bei Qualitäten nur von „ähnlich“ (ὁμοίον) spricht. Außerdem „nehmen sie kein mehr oder weniger an“ (6a 17): ein Maßausdruck kann nicht durch „mehr“ erweitert werden: Größenangaben sind absolut, nicht graduierbar und nicht in Vergleichen verwendbar (6a 18).[13]

Die genannten Größen gelten als die einzigen echten Quantitäten (Kap.6, 5a38); es werden aber auch indirekte Quantitäten anerkannt, etwa die Arbeit, deren Maß die Zeit zu ihrer Ausführung ist.

Aristoteles geht nicht darauf ein, wie die Maßausdrücke in den Aussagen $x(y)$ und $x[y]$ vorkommen. Man müßte sie wohl in die Prädikativsätze $x[y]$, d.h. „y ist x“ mit Adjektiv x, einfügen: „y ist Q x“, etwa „y ist 20 m^2 groß“.

Von diesen *bestimmten* Größenangaben unterscheidet Aristoteles die *unbestimmten* und *relativen* Größenangaben wie „groß“ und „klein“, ebenso „viel“ und „wenig“. Dies sind keine Quantitäten, sondern Vergleiche mit einem impliziten Bezugsobjekt: der Berg ist klein im Vergleich zu anderen Bergen. Der Maßstab hängt von der Art der Objekte ab: das große Hirsekorn ist ein kleiner Gegenstand. Solche Adjektive drücken *relative* Eigenschaften aus.

[13]Wenn man Vergleichsaussagen ausschließt, bleibt unklar, wie über die kontinuierlichen Größen gesprochen wird. Die Entdeckung der kontinuierlichen Größen beruhte doch gerade auf der Inkommensurabilität, daß die Länge der Diagonale eines Quadrats *kein* Vielfaches der Seitenlänge (Maßeinheit) ist.

2.1.3 Weitere Kategorien

Auf die anderen Kategorien bei Aristoteles, und wie man sie interpretieren könnte, wird hier nicht eingegangen. Sie sind im Original weit weniger klar ausgeführt, werden aber dort deutlich im Hinblick auf Ausdrucksweisen des Griechischen diskutiert, grob wie folgt:

Relativa: Relationen und Vergleichsausdrücke

Beschaffenheit (Kap.8): Adjektive

Tun und Erleiden (Kap.9, S.67): Aktiv- und Passiv bei Verben

Gegensätze (Kap.10, S.67–83, und 11, S.83–85): Negationspartikel

Früher und Zugleich (Kapitel 12, S.85, und 13, S.87-89): Zeitadverbien

Veränderung (Kapitel 14, S. 89–93):

Haben (Kapitel 15, S.93): Verben des Besitzes, Zustandsverben

Eine Darstellung der Grundideen und der Versuch einer abstrakten Rekonstruktion dieser Kategorien wäre wünschenswert und instruktiv, ist aber im Rahmen des Proseminars nicht zu machen.

2.2 Interpretation des Kategoriensystems von Aristoteles

Vielleicht kann man das System des Aristoteles grob wie folgt verstehen:

Man hat einerseits *Objekte*, und andererseits *Prädikate*, die von Objekten ausgesagt werden. Bei den Prädikaten gibt es

(a) Klassifikationsprädikate, oder *Typen* σ, die eine partiell geordnete Menge von Gattungen bilden; die Gattungen werden durch Unterscheidungsprädikate $U : \sigma \to \mathbb{B}$ in disjunkte Arten zerlegt, wobei die Anzahl der Unterscheidungsprädikate und damit die der Arten endlich ist.

    Jedes Objekt gehört zu genau einer Art. Die partielle Ordnung $\leq$ der Gattungen ist mit der Zugehörigkeit der Objekte zu den Arten verträglich: falls $U : \sigma \to \mathbb{B}$ und $\sigma \leq \tau$, so folgt aus $U(a) = 1$, daß $a : \sigma$ und daraus wiederum, daß $a : \tau$.

    Die Aussagen $U(a)$ haben boole'sche Werte, die nicht von der Zeit (und anderen Umständen) abhängen.

(b) Attribute: diese sind im allgemeinen von der Zeit abhängig und teffen in unterschiedlichem Ausmaß auf Objekte zu, d.h. ihr Wertebereich ist diskret oder kontinuierlich geordnet.

(c) Relationen: Prädikate, die Beziehungen zwischen mehreren Objekten aussagen, oder Werte eines Attributs von Objekten vergleichen, und funktional Zusammenhänge oder Abhängigkeiten zwischen Objekten ausdrücken. (Einen Funktionsbegriff hat Aristoteles aber noch nicht.)

Die Elementaraussagen $x(y)$ sind Klassifizierungsaussagen $y : x$ wo x ein Typ oder ein Unterscheidungsprädikat U (oder eine Art, d.h. eine boole'sche Kombination davon); in diesem Fall ist ihr Wert $[\![x(y)]\!] \in \mathbb{B}$ von der Zeit unabhängig.

Die Elementaraussagen $x[y]$ sind Aussagen über Attribute von Objekten; in diesem Fall ist ihr Wert $[\![x[y]]\!]_t \in \mathbb{B}$ von der Zeit t und anderen Umständen der Äußerung abhängig, und die Aussageform $x[y]$ ist als Abkürzung zu verstehen: etwa als „das Objekt y hat zur Zeit t bezüglich des Attributs A den Wert x“, oder $A(y)(t) = x$, wobei das Attribut A sich implizit aus seinem Wert x ergibt.

Da Attribute A i.a. Werte $x \in |A|$ auf einer Skala $(|A|, <_A)$ annehmen, lassen sich die Attributwerte desselben Objekts y zu verschiedenen Zeiten t, t' oder verschiedener Objekte y, y' miteinander vergleichen:

$$[\![y <_A y']\!]_t := A(y)(t) <_A A(y')(t)$$

oder (mit kruder Formalisierung)

$$[\![y <_A t']\!]_t := A(y)(t) <_A A(y)(t').$$

Das müßte eine (implizit angenomme) Art sein, wie Relationen über Gradaussagen und Komparative von Adjektiven aus Attributaussaben aufgebaut werden können; andernfalls muß man eine weitere Form von Grundaussagen annehmen.

Vielleicht sollen Aussagen der Form „das Objekt y hat die Eigenschaft A“ als Abkürzungen interpretiert werden, im Sinne von

$$\begin{aligned} & [\![y \text{ hat die Eigenschaft } A]\!] \\ = \; & [\![y \text{ hat die Eigenschaft } A \text{ zur Zeit } t \text{ am Ort } O \text{ im Maße } x]\!] \\ = \; & [\![x[y]]\!]_{t,O} \in \mathbb{B}. \end{aligned}$$

Ob es allerdings praktikabel ist, alle Aussagen auf die Form von Grundprädikaten $P(y, t, O, x, \ldots)$ mit Argumenten für Zeit, Ort, Modus usw. zurückzuführen, ist fraglich. Zumindest müssen dazu Zeitpunkte, Orte, Skalenwerte u.a. wie Objekte behandelt werden. Das wollte Aristoteles vermutlich nicht.

Vielleicht ist es eher umgekehrt: die Kategorien sind eine Einteilung der Prädikate nach verschiedenen semantischen Dimensionen, und jedes Objekt y hat bezüglich jeder Dimension X einen Wert $X(y)$:

Attribute X

Dimension	Werte
Substanz	Lebewesen, Mensch, ...
Tätigkeit	gehen, sprechen, ...
Ort	in Athen, im Lyzeum, ...
Eigenschaft	
(Farbe, Alter, ...)	grün, 20 Jahre, ...
Zeit	jetzt, ...
...	...

Dann macht allerdings die Kategorie Zeit Schwierigkeiten, da sie kaum den Objekten zugeordnet ist (es sei denn, man meint die Lebenszeit).

Bem. Bei den Quantitäten scheint es darum zu gehen, daß man in Aussagen der Form „das Objekt y hat bezüglich des Attributs A den Wert x" einen Funktionswert angibt:

$$[\![xA[y]]\!]_t := \begin{cases} 1, & A(y)(t) = x \\ 0, & \text{sonst} \end{cases}$$

Da in manchen Fällen das Attribut nicht genannt wird, könnte man diese Aussagen als Spezialfall der Prädikativsätze $x[y]$ verstehen, im Sinne von „die Strecke y ist x (lang)". Syntaktisch gesehen wären die Quantitäten dann Maßausdrücke x, die mit einem (relativen) Adjektiv wie „lang" ein einstelliges Prädikat „x lang sein" bilden.

3 Auswertung von Ausdrücken

3.1 Die Syntax einer Sprache

Die *Syntax* einer Sprache L legt fest, was die erlaubten Ausdrücke der Sprache sind. Die Gesamtheit der Ausdrücke wird in verschiedene *syntaktische Kategorien* eingeteilt. Bei formalen Sprachen erfolgt diese Einteilung nur nach der Art des Aufbaus der Ausdrücke, bei natürlichen Sprachen spielt die Funktion eines Ausdrucks im übergeordneten Ausdruck auch eine Rolle: Ausdrücke unterschiedlicher Bauart können eine adverbiale Funktion im Satz oder eine determinierende Funktion in einem Nominalausdruck erfüllen.

Wir betrachten hier nur Sprachen, bei denen die syntaktischen Kategorien die Ausdrücke nur nach ihrer Bauart unterscheiden.

Definition 3.1 Die Syntax einer Sprache wird durch eine *Grammatik* festgelegt, die

(a) die syntaktischen Kategorien τ (Arten von Ausdrücken) angibt und unter ihnen (meist) eine Hauptkategorie, die der Sätze, auszeichnet,

(b) für jede syntaktische Kategorie τ eine Reihe von Regeln zur Konstruktion von Ausdrücken dieser Kategorie (Grammatikregeln) angibt; und zwar

  (a) zum einen die einfachen oder atomaren Ausdrücke der Kategorie τ,

  (b) zum anderen die (aus Ausdrücken i.a. verschiedener Kategorien) zusammengesetzten Ausdrücke der Kategorie τ.

Erlaubte Ausdrücke der Sprache sind alle Ausdrücke, die nach den Regeln konstruiert werden können.[14]

Die Grammatikregeln können wir in den meisten Fällen im Format der kontextfreien Grammatiken angeben. Dabei schreibt man alle k Regeln, mit denen Ausdrücke t der Kategorien τ konstruiert werden, als eine Alternative

$$\tau ::= \alpha_1 \mid \ldots \mid \alpha_k,$$

wobei die α_i die einzelnen Konstruktionsmuster sind, d.h. Folgen aus atomaren Ausdrücken oder Kategorien (die für beliebige Ausdrücke dieser Kategorie stehen – an ihrer Stelle benutzt man daher oft auch eine Metavariable für Ausdrücke dieser Kategorie.)

Wir illustrieren das an vier (vereinfachten) Beispielsprachen: einer Sprache *PL* der Prädikatenlogik, der Sprache *KL* der klassischen Logik, einer natürlichen Sprache *NL*, und einer Programmiersprache *ProgL*.

[14] Da wir uns nur mit der Schriftsprache beschäftigen, sind für uns alle Ausdrücke aus Zeichen eines Alphabets zusammengesetzt. Normalerweise sind es einfach Zeichenreihen; es können aber auch Tupel von Zeichenreihen sein, wenn man unzusammenhängende Ausdrücke braucht.

3.1.1 Die Sprache der Prädikatenlogik: PL

Beispiel 3.2 Für eine Sprache PL der Prädikatenlogik (ohne Quantoren) braucht man folgende syntaktischen Kategorien und Ausdrücke:

Synt. Kategorie		Metavariable und Konstruktionsmöglichkeiten	
Funktor	F	$f ::=$	$f_1 \mid f_2 \mid \dots$
Prädikat	P	$\rho ::=$	$P_1 \mid P_2 \mid P_3 \mid \cdots$
Junktor	J	$o ::=$	$\neg \mid \wedge \mid \vee \mid \rightarrow \mid \leftrightarrow$
Term	T	$t ::=$	$f^{(n)}(t_1, \dots, t_n)$
Formel	Fml	$\varphi ::=$	$\rho^{(n)}(t_1, \dots, t_n) \mid \neg\varphi \mid (\varphi_1\ o^{(2)}\ \varphi_2)$

Hier enthalten alle Kategorien außer den Termen und Formeln nur einfache Ausdrücke, und alle Formeln sind zusammengesetzte Ausdrücke. (Die einfachen Ausdrücke haben (implizit) eine Stelligkeit $n \in \mathbb{N}$, die in der Form $f^{(n)}, P^{(n)}, o^{(n)}$ angegeben wird, wenn das wichtig erscheint.)

3.1.2 Die Sprache der Klassischen Logik: KL

Beispiel 3.3 Die Sprache KL der klassischen Logik des Aristoteles verwendet andere Kategorien und Ausdrücke:

Synt. Kategorie	Metavariable und Konstruktionsmöglichkeiten	
Terminus	$\tau ::=$	$A \mid B \mid C \mid D \mid \dots$
Kopula	$\kappa ::=$	$a \mid e \mid i \mid o$ (alle, keine, einige, nicht alle)
einf. Aussage	$\varphi ::=$	$(\tau_1\ \kappa\ \tau_2)$
zusm. Aussage	$S ::=$	$(\varphi_1 \rightarrow \varphi_0) \mid (\varphi_1 \wedge \varphi_2 \rightarrow \varphi_0)$

Die einfachen Aussagen liest man (und interpretiert sie mit Mengen) wie

$A\ a\ B$	alle A sind B			$(A \subseteq B)$
$A\ e\ B$	keine A sind B	=	alle A sind keine B	$(A \subseteq \overline{B})$
$A\ i\ B$	einige A sind B			$(A \not\subseteq \overline{B})$
$A\ o\ B$	nicht alle A sind B	=	einige A sind keine B	$(A \not\subseteq B)$

Aristoteles interessierte sich für *Syllogismen*, d.h. für diejenigen zusammengesetzten Aussagen, die (bei der angedeuteten Interpretation) wahr sind.

3.1.3 Eine einfaches Fragment einer natürlichen Sprache: NL

Beispiel 3.4 Ein einfaches Fragment NL einer natürlichen Sprache hat etwa die folgenden syntaktischen Kategorien und Ausdrücke:

Syntaktische Kategorie		Metavariable und Konstruktionsmöglichkeiten	
Eigenname	PN	$e ::=$	*Anna* | *Emil* | *Maria* | ...
Gemeinname	N	$\alpha ::=$	*Frau* | *Mann* | *Student* | ...
Artikel	DET	$\delta ::=$	*die* | *der*
Verb	V	$\rho ::=$	$arbeitet^{(1)} \mid kennt^{(2)} \mid liebt^{(2)} \mid \ldots$
Konjunktion	$CONJ$	$o ::=$	$nicht^{(1)} \mid und^{(2)} \mid oder^{(2)}$
Kopula	COP	$\kappa ::=$	*ist*
Nominalausdruck	NP	$\tau ::=$	$e \mid \delta\ \alpha$
Satz	S	$\varphi ::=$	$\tau\ \kappa\ \tau \mid \tau\ \rho^{(1)} \mid \tau_1\ \rho^{(2)}\ \tau_2$
		$\mid$	$\tau_1\ \kappa\ o^{(1)}\ \tau_2 \mid \tau\ \rho^{(1)}\ o^{(1)} \mid \tau_1\ \rho^{(2)}\ \tau_2\ o^{(1)}$
		$\mid$	$\varphi_1\ o^{(2)}\ \varphi_2$

Damit lassen sich z.B. die Sätze bilden: *Emil ist nicht der Student*, *der Mann kennt die Frau*, *der Student arbeitet und Maria liebt Emil nicht*.

Die Satzregeln schreibt man oft mit den Kategorienbezeichnungen, z.B.

$$S \quad ::= \quad PN\ \textit{ist}\ NP \mid NP\ V \mid NP_1\ V\ NP_2 \mid \ldots$$

Es ist klar, daß die Regeln zu grob sind, weil keine Rücksicht auf morphologische Formvarianten der Wörter genommen wird. Daher sind z.B. *die Student* und *der Frau* Nominalausdrücke dieser Beispielsprache NL.

3.1.4 Eine imperative Programmiersprache: WL

Beispiel 3.5 (Programmiersprache) Wir betrachten eine imperative Programmiersprache mit folgenden syntaktischen Kategorien und Ausdrücken:

Synt. Kategorie	Metavariable und Konstruktionsmöglichkeiten		
Variable	$x ::=$	$x_1 \mid x_2 \mid x_3 \mid \ldots$	
Term	$t ::=$	$x \mid 0 \mid 1 \mid (t_1 + t_2) \mid (t_1 - t_2)$	
Test	$\tau ::=$	$t_1 = t_2 \mid t_1 < t_2$	
	$\mid$	$(\tau_1 \wedge \tau_2) \mid (\tau_1 \vee \tau_2)$	
Anweisung	$\alpha ::=$	$(x \leftarrow t)$	(Zuweisung)
	$\mid$	$(\alpha_1; \alpha_2)$	(Nacheinanderausführung)
	$\mid$	$(\textit{if}\ \tau\ \textit{then}\ \alpha_1\ \textit{else}\ \alpha_2)$	(Fallunterscheidung)
	$\mid$	$(\textit{while}\ \tau\ \textit{do}\ \alpha)$	(bedingte Wiederholung)

Wir benutzen die übliche Schreibweise von Zahlen, z.B. 4 statt des umständlichen Terms $(1 + (1 + (1 + 1)))$. Damit kann man Programme wie

$$x \leftarrow 2\,;\,(\mathit{while}\ x < 7\ \mathit{do}\ (x \leftarrow y + x\,;\, y \leftarrow y - 1))$$

schreiben.

3.2 Interpretation und Auswertung von Ausdrücken

Definition 3.6 Eine (denotationelle) *Interpretation* (oder eine L-Struktur)

$$\mathcal{D} = (D_\tau, \ldots, c^{\mathcal{D}}, \ldots)$$

für eine Sprache L gibt Wertebereiche D_τ für die verschiedenen syntaktischen Kategorien τ und Bedeutungen $c^{\mathcal{D}} \in D_\tau$ für die einfachen Ausdrücke $c : \tau$ von L an.

Eine *Auswertungsfunktion* $[\![\cdot]\!]^{\mathcal{D}}$ berechnet die Bedeutungen zusammengesetzter Ausdrücke aus den Bedeutungen der in ihnen vorkommenden einfachen Ausdrücke. Auch für zusammengesetzte Ausdrücke $t : \tau$ soll $[\![t]\!]^{\mathcal{D}} \in D_\tau$ gelten.

Wenn die Ausdrücke t Variable $x : \tau$ enthalten, für die $\mathcal{D}$ keine Bedeutung angibt, braucht die Auswertungsfunktion *Belegungen* $h_\tau : Var_\tau \to D_\tau$, um den Wert $[\![t]\!]h^{\mathcal{D}}$ mit Hilfe der $h = \langle h_\tau \mid \tau \text{ eine Kategorie}\rangle$ berechnen zu können.

Wir geben nun für die vier Beispielsprachen jeweils eine Interpretation $\mathcal{D}$ und eine Auswertungsfunktion $[\![\cdot]\!]^{\mathcal{D}}$ zur Auswertung der Ausdrücke der Sprache bei dieser Interpretation $\mathcal{D}$ an.

3.2.1 Interpretation von PL

Beispiel 3.7 (Forts. von Beispiel 3.2) Eine Interpretation $\mathcal{D}$ der Sprache PL muß zuerst die Wertebereiche der syntaktischen Kategorien angeben.

(a) Werte der Terme stammen aus dem *Individuenbereich* oder *Diskursuniversum* der Objekte, über die gesprochen wird, einer Menge

$$D_T \neq \emptyset. \tag{1}$$

(b) Werte der Formeln sind die *Wahrheitswerte* 0 (*falsch*) und 1 (*wahr*), daher

$$D_{Fml} := \{0, 1\} =: \mathbb{B}. \tag{2}$$

(c) Werte der Funktoren sind *Zuordnungsfunktionen zwischen Individuen*:

$$D_F \subseteq \{f \mid n \in \mathbb{N}, f : D_T^n \to D_T\}. \tag{3}$$

(d) Werte der Prädikate sind *Testfunktionen auf Individuen*:

$$D_P \subseteq \{p \mid n \in \mathbb{N}, p : D_T^n \to \mathbb{B}\}. \tag{4}$$

(e) Werte der Junktoren sind *Wahrheitsfunktionen*:

$$D_J \subseteq \{f \mid n \in \mathbb{N}, f : \mathbb{B}^n \to \mathbb{B}\}. \tag{5}$$

Weiterhin muß $\mathcal{D}$ die Werte der einfachen Ausdrücke jeder Kategorie angeben. Als Werte der Junktoren $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ nimmt man die durch die üblichen Funktionstabellen festgelegten Wahrheitsfunktionen $\neg^{\mathbb{B}}, \wedge^{\mathbb{B}}, \vee^{\mathbb{B}}, \rightarrow^{\mathbb{B}}, \leftrightarrow^{\mathbb{B}}$, die durch die folgenden Tabellen festgelegt sind:

$\neg^{\mathbb{B}}$	
0	1
1	0

$\wedge^{\mathbb{B}}$	0	1
0	0	0
1	0	1

$\vee^{\mathbb{B}}$	0	1
0	0	1
1	1	1

$\rightarrow^{\mathbb{B}}$	0	1
0	1	1
1	0	1

$\leftrightarrow^{\mathbb{B}}$	0	1
0	1	0
1	0	1

Für die Funktoren $f_i^{(n)}$ muß $\mathcal{D}$ eine Zuordnungsfunktion $f_i^{\mathcal{D}} : D_T^n \rightarrow D_T$ aus D_F und für die Prädikate $P_i^{(n)}$ eine Testfunktion $P_i^{\mathcal{D}} : D_T^n \rightarrow \mathbb{B}$ aus D_P angeben.

Der Wert $[\![\alpha]\!]^{\mathcal{D}}$ eines Ausdrucks α wird dann durch Induktion über den Aufbau der Ausdrücke definiert: f_i, P_i, o,

(a) $[\![f_i]\!] := f_i^{\mathcal{D}}$ für Funktoren f_i,

(b) $[\![P_i]\!] := P_i^{\mathcal{D}}$ für Prädikate P_i

(c) $[\![o_i]\!] := o_i^{\mathbb{B}}$, für Junktoren o_i,

(d) $[\![f_i(t_1, \ldots, t_n)]\!] := [\![f_i]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$ für Terme $f_i(t_1, \ldots, t_n)$,

(e) $[\![P_i(t_1, \ldots, t_n)]\!] := [\![P_i]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$ für Formeln $P_i(t_1, \ldots, t_n)$, sowie

$$[\![\neg\varphi]\!] := \neg^{\mathbb{B}}([\![\varphi]\!]) \quad \text{und} \quad [\![(\varphi_1 \; o \; \varphi_2)]\!] := o^{\mathbb{B}}([\![\varphi_1]\!], [\![\varphi_2]\!])$$

für die mit ein- bzw. zweistelligem Junktor zusammengesetzten Formeln.

Man rechne nach, daß tatsächlich $[\![\alpha]\!]^{\mathcal{D}} \in D_\tau$ ist, wenn α ein Ausdruck der Kategorie τ ist.

Bemerkung: Für 0-stellige Funktoren $f_i^{(0)}$ schreibt man den Term $f_i()$ einfach als f_i und bezeichnet ihn als *Individuenkonstante.* Die 0-stellige Zuordnungsfunktion $f_i^{\mathcal{D}} : D_T^0 \rightarrow D_T$ ordnet dem leeren Tupel $()$, dem einzigen Element von D_T^0, ein Individuum $f_i^{\mathcal{D}}() \in D_T$ zu; man faßt $f_i^{\mathcal{D}}()$ als Wert der Individuenkonstante f_i auf.

Bemerkung 3.8 Wir haben Prädikate $P_i^{(n)}$ durch Test- oder *charakteristische* Funktionen $\chi : D_T^n \rightarrow \mathbb{B}$ interpretiert, nicht durch n-stellige Relationen $R \subseteq D_T^n$. Natürlich entspricht jeder Testfunktion χ eine Relation R_χ, nämlich

$$R_\chi := \{(a_1, \ldots, a_n) \in D_T^n \mid \chi(a_1, \ldots, a_n) = 1\}.$$

Umgekehrt entspricht jeder Relation R eine Testfunktion χ_R, die durch

$$\chi_R(a_1, \ldots, a_n) := \begin{cases} 1, & \text{falls } (a_1, \ldots, a_n) \in R \\ 0, & \text{sonst} \end{cases}$$

gegeben ist. Mathematisch kann man es so machen, wie es einem besser gefällt. Aber für die Programmierung besteht ein großer Unterschied: ist die Relation R gebeben, z.B. als Liste von Tupeln, so kann man leicht ein Programm für χ_R schreiben: es durchsucht einfach die Liste nach dem Argument $(a_1, \ldots, a_n)$. Ist aber umgekehrt ein Programm χ gegeben, so kann man die Relation R_χ nur konstruieren, wenn D_T endlich ist: man muß ja für jedes $(a_1, \ldots, a_n) \in D_T^n$ mit χ testen, ob es in die Liste R_χ aufgenommen werden muß.

Für die Auswertung der Formel $P_i(t_1, \ldots, t_n)$ braucht man aber nur die Testfunktion $P_i^{\mathcal{D}}$ zu haben, nicht die zugehörige Relation.

3.2.2 Interpretation von KL

Anders sieht eine Interpretation der klassischen Logik aus:

Beispiel 3.9 (Forts. von Beispiel 3.3) Die normale Interpretation $\mathcal{D}$ der klassischen Logik benutzt folgende Wertebereiche:

(a) Werte der Terme sind *Mengen von Individuen* eines Diskursuniversums U,

$$D_\tau \subseteq 2^U := \{M \mid M \subseteq U\}. \tag{6}$$

(b) Wert einer Kopula ist eine 2-stellige *Testfunktion auf Individuenmengen*,

$$D_\kappa \subseteq \{f \mid f : D_\tau^2 \to \mathbb{B}\} \tag{7}$$

(c) Wert von Aussagen sind die Wahrheitswerte,

$$D_S := \mathbb{B}. \tag{8}$$

Die Interpretation der vier Kopulae sind –mit $\overline{B} := U \setminus B$–

$$a^{\mathcal{D}}(A,B) := \begin{cases} 1, & \text{falls } A \subseteq B \\ 0, & \text{sonst} \end{cases}, \qquad o^{\mathcal{D}}(A,B) := \begin{cases} 1, & \text{falls } A \not\subseteq B \\ 0, & \text{sonst} \end{cases}$$

$$i^{\mathcal{D}}(A,B) := \begin{cases} 1, & \text{falls } A \not\subseteq \overline{B} \\ 0, & \text{sonst} \end{cases}, \qquad e^{\mathcal{D}}(A,B) := \begin{cases} 1, & \text{falls } A \subseteq \overline{B} \\ 0, & \text{sonst} \end{cases}$$

Die Auswertung wird durch Induktion über den Aufbau der Aussagen definiert:

$$[\![(A\ \kappa\ B)]\!] := [\![\kappa]\!]([\![A]\!], [\![B]\!]),$$

und mit den Wahrheitsfunktionen für die zusammengesetzten Aussagen, wie

$$[\![(\varphi_1 \to \varphi_2)]\!] := \to^{\mathbb{B}} ([\![\varphi_1]\!], [\![\varphi_2]\!]).$$

Beachte, daß der Quantor der natürlichsprachlichen Aussage *alle A sind B* in der Kopula a von $(A\ a\ B)$ enthalten ist; analog bei e, i, o. (In Phrasenstrukturgrammatiken wie NL sind die Quantoren in den Nominalphrasen enthalten.)

3.2.3 Interpretation von NL

Beispiel 3.10 (Forts. von Beispiel 3.4) Eine Interpretation $\mathcal{D}$ der Sprache NL kann die Wertebereiche der Kategorien und die Werte der einfachen Ausdrücke wie folgt festlegen:

(a) Werte der Sätze sind die Wahrheitswerte: $D_S := \{0,1\} = \mathbb{B}$.

(b) Werte der Konjunktionen sind ein- oder zweistellige Wahrheitsfunktionen,

$$D_{CONJ} \subseteq \{f \mid 1 \leq n \leq 2, f : \mathbb{B}^n \to \mathbb{B}\}, \tag{9}$$

darunter $nicht^{\mathcal{D}} := \neg^{\mathbb{B}}$, $und^{\mathcal{D}} := \wedge^{\mathbb{B}}$, und $oder^{\mathcal{D}} := \vee^{\mathbb{B}}$.

(c) Werte der Eigennamen sind die –sagen wir– 23 Personen im Hörsaal,

$$D_{PN} := \{a,b,c,d,e,f,\ldots,u,v,w\}, \tag{10}$$

die durch die Anfangsbuchstaben ihrer Vornamen dargestellt sind, davon

$$Anna^{\mathcal{D}} := a, \quad Emil^{\mathcal{D}} := e, \quad Maria^{\mathcal{D}} := m.$$

(d) Werte der Gemeinnamen sind einstellige *Testfunktionen auf Personen*,

$$D_N \subseteq \{\chi \mid \chi : D_{PN} \to \mathbb{B}\}, \tag{11}$$

darunter seien die Testfunktionen [15]

$$\begin{aligned} Frau^{\mathcal{D}} &:= \chi_{\{a,b,c,g,h,i,m,n,o,s,t,u\}}, \\ Mann^{\mathcal{D}} &:= \chi_{\{d,e,f,j,k,l,p,q,r,v,w\}}, \\ Student^{\mathcal{D}} &:= \chi_{\overline{\{u,v,w\}}}, \end{aligned}$$

also z.B. seien alle Personen außer u, v, w Studenten.

(e) Werte der Verben sind ein- oder mehrstellige *Testfunktionen auf Personen*,

$$D_V \subseteq \{f \mid n \in \mathbb{N}, f : D_{PN}^n \to \mathbb{B}\}, \tag{12}$$

darunter

$$\begin{aligned} arbeitet^{\mathcal{D}} &:= \chi_{\{a,b,c,d,e\}}, \\ kennt^{\mathcal{D}} &:= \chi_{\{(a,e),(a,m),(e,a),(m,e)\}\cup\{(x,y)\mid x=y\}} \\ liebt^{\mathcal{D}} &:= \chi_{\{(e,m),(a,e)\}} \end{aligned}$$

[15] Beachte, daß es "realistischer" ist, die Gemeinnamen durch Testfunktionen χ als durch die entsprechenden Mengen $R_\chi = \{a \in D_{PN} \mid \chi(a) = 1\}$ darzustellen: was "Student" bedeutet, weiß man, wenn man Studenten (z.B. an ihrem Ausweis) erkennen kann – dazu braucht man die Gesamtheit aller Studenten nicht.

(f) Werte der Artikel sind *Auswahlfunktionen*, die zu einer einstelligen Testfunktion auf Personen eine Person auswählen, die den Test besteht:

$$D_{DET} \subseteq \{g : (D_{PN} \to \mathbb{B}) \to D_{PN} \mid g(f) \in \{x \mid f(x) = 1\}\}, \tag{13}$$

darunter $die^{\mathcal{D}} := der^{\mathcal{D}} := g$ für die wie folgt definierte Funktion:

$$g(f) \quad := \quad \begin{cases} \text{das bzgl. (10) erste } x \text{ mit } f(x) = 1, & \text{falls es das gibt,} \\ a, & \text{sonst.} \end{cases}$$

(Diese Interpretation ist nur für *bestimmte* Artikel gedacht; für indefinite Artikel und Quantoren wird später eine andere Interpretation entwickelt.)

(g) Wert der Kopula *ist* ist der *Gleichheitstest* auf Individuen, die durch

$$ist^{\mathcal{D}}(x, y) := \begin{cases} 1, & \text{falls } x = y \\ 0, & \text{sonst} \end{cases} \tag{14}$$

definierte zweistellige Testfunktion auf Individuen.[16]

(h) Werte der Nominalausdrücke sind Individuen,

$$D_{NP} = D_{PN} \tag{15}$$

Die Auswertungsfunktion $[\![\cdot]\!]^{\mathcal{D}}$ ist für einfache Ausdrücke $\alpha : \sigma$ durch

$$[\![\alpha]\!] := \alpha^{\mathcal{D}} \in D_\sigma$$

definiert und für die zusammengesetzten induktiv gemäß folgender Fälle:[17]

(a) $[\![\delta\ \alpha]\!] := [\![\delta]\!]([\![\alpha]\!])$,

(b) $[\![\tau_1\ \mathit{ist}\ \tau_2]\!] := [\![\mathit{ist}]\!]([\![\tau_1]\!], [\![\tau_2]\!])$,

(c) $[\![\tau\ \rho]\!] := [\![\rho]\!]([\![\tau]\!])$,

(d) $[\![\tau_1\ \rho\ \tau_2]\!] := [\![\rho]\!]([\![\tau_1]\!], [\![\tau_2]\!])$,

(e) $[\![\tau\ \rho\ \mathit{nicht}]\!] := \neg^{\mathbb{B}}([\![\tau\ \rho]\!])$,

(f) $[\![\tau_1\ \mathit{ist\ nicht}\ \tau_2]\!] := \neg^{\mathbb{B}}([\![\tau_1\ \mathit{ist}\ \tau_2]\!])$,

(g) $[\![\tau_1\ \rho\ \tau_2\ \mathit{nicht}]\!] := \neg^{\mathbb{B}}([\![\tau_1\ \rho\ \tau_2]\!])$,

(h) $[\![\varphi_1\ o\ \varphi_2]\!] := o^{\mathbb{B}}([\![\varphi_1]\!], [\![\varphi_2]\!])$.

[16] Diese Definition ist nur für *eine* der Verwendungen der Kopula *ist* sinnvoll!

[17] Wir schreiben nur noch $[\![\alpha]\!]$ statt $[\![\alpha]\!]^{\mathcal{D}}$.

Die Auswertung ergibt, daß bei dieser Interpretation der Satz *Emil ist nicht der Student* wahr wird:

$$\begin{aligned}
&[\![\textit{Emil ist nicht der Student}]\!]\\
&=_{(f)} \quad \neg^{\mathbb{B}}([\![\textit{Emil ist der Student}]\!])\\
&=_{(b)} \quad \neg^{\mathbb{B}}([\![\textit{ist}]\!]([\![\textit{Emil}]\!], [\![\textit{der Student}]\!]))\\
&= \quad \neg^{\mathbb{B}}([\![\textit{ist}]\!](\textit{Emil}^{\mathcal{D}}, [\![\textit{der Student}]\!]))\\
&= \quad \neg^{\mathbb{B}}([\![\textit{ist}]\!](e, [\![\textit{der Student}]\!]))\\
&=_{(a)} \quad \neg^{\mathbb{B}}([\![\textit{ist}]\!](e, [\![\textit{der}]\!]([\![\textit{Student}]\!])))\\
&= \quad \neg^{\mathbb{B}}([\![\textit{ist}]\!](e, \textit{der}^{\mathcal{D}}(\chi_{\overline{\{u,v,w\}}})))\\
&= \quad \neg^{\mathbb{B}}(\textit{ist}^{\mathcal{D}}(e, \textit{erstes Element von } \{a, b, \dots, r, s, t\}))\\
&= \quad \neg^{\mathbb{B}}(\textit{ist}^{\mathcal{D}}(e, a))\\
&= \quad \neg^{\mathbb{B}}(0)\\
&= \quad 1.
\end{aligned}$$

Aufgabe 3.1 (a) Berechne den Wert $[\![\textit{Der Mann kennt die Frau}]\!]^{\mathcal{D}}$.

(b) Zeige, daß in der Tat $[\![\cdot]\!]^{\mathcal{D}}$ eine Auswertungsfunktion ist, d.h. daß für jeden Ausdruck α einer Kategorie σ tatsächlich $[\![\alpha]\!]^{\mathcal{D}} \in D_\sigma$ ist.

3.2.4 Interpretation von WL

Zum Schluß interpretieren wir die Programmiersprache WL der *while*-Ausdrücke.

Beispiel 3.11 (Forts. von Beispiel 3.5) Unsere Interpretation $\mathcal{D}$ der Programme von WL soll den Kategorien folgende Wertebereiche zuordnen:

(a) Werte der Variablen und Terme seien ganze Zahlen:

$$D_{\textit{Var}} = D_{\textit{Term}} := \mathbb{Z} \tag{16}$$

(b) Werte der Tests sind Wahrheitswerte:

$$D_{\textit{Test}} := \mathbb{B}. \tag{17}$$

Werte der Anweisungen sind sicher weder Wahrheitswerte noch Objekte, aber was dann? Wir verstehen die Programmvariablen als Speicheradressen, bei denen während der Ausführung des Programms Zahlen abgelegt und geholt werden. Im Fall einer Zuweisung $x \leftarrow t$ soll die bei x gespeicherte Zahl durch den Wert des Terms t ersetzt werden.

Ein *Speicherzustand* ist eine Belegung $h : \textit{Var} \to \mathbb{Z}$ der Variablen des Programms, und $h(x)$ die „bei x gespeicherte“ Zahl. Als Bedeutung einer Anweisung

α können wir dann den durch Ausführen von α erreichten Speicherzustand verstehen:[18]

(c) Werte der Anweisungen sind Speicherzustände:

$$D_{Anweisung} := S \qquad \text{mit } S := \{h \mid h : Var \to D_{Var}\}. \tag{18}$$

Da Terme t und Anweisungen α Variable enthalten, hängen ihre Werte von den Werten $h(x)$ der in ihnen vorkommenden Variablen x ab, d.h. von dem Speicherzustand h, der bei der Auswertung vorliegt. Die Auswertung ist also *relativ* zu einem Speicherzustand h, sodaß wir $[\![\cdot]\!]^{\mathcal{D}}_h$ (statt $[\![\cdot]\!]^{\mathcal{D}}$) definieren:

(a) $[\![x]\!]_h := h(x)$,

(b) $[\![0]\!]_h := 0^{\mathbb{Z}}, \qquad [\![1]\!]_h := 1^{\mathbb{Z}}$,

(c) $[\![t_1 + t_2]\!]_h := [\![t_1]\!]_h +^{\mathbb{Z}} [\![t_2]\!]_h, \qquad [\![t_1 - t_2]\!]_h := [\![t_1]\!]_h -^{\mathbb{Z}} [\![t_2]\!]_h$,

(d) $[\![t_1 = t_2]\!]_h := \begin{cases} 1, & \text{falls } [\![t_1]\!]_h = [\![t_2]\!]_h, \\ 0, & \text{sonst} \end{cases},$

(e) $[\![t_1 < t_2]\!]_h := \begin{cases} 1, & \text{falls } [\![t_1]\!]_h <^{\mathbb{Z}} [\![t_2]\!]_h, \\ 0, & \text{sonst,} \end{cases}$

(f) $[\![\tau_1 \wedge \tau_2]\!]_h := [\![\tau_1]\!]_h \wedge^{\mathbb{B}} [\![\tau_2]\!]_h, \qquad [\![\tau_1 \vee \tau_2]\!]_h := [\![\tau_1]\!]_h \vee^{\mathbb{B}} [\![\tau_2]\!]_h$,

(g) $[\![x \leftarrow t]\!]_h(y) := \begin{cases} [\![t]\!]_h, & \text{falls } y = x, \\ h(y), & \text{sonst,} \end{cases},$

d.h. nach Ausführen der Zuweisung $x \leftarrow t$ liegt derselbe Speicherzustand wir vorher vor, außer daß bei x der Wert von t gespeichert ist,

(h) $[\![(\alpha_1\, ;\, \alpha_2)]\!]_h := [\![\alpha_2]\!]_{[\![\alpha_1]\!]_h}$,

d.h. der Speicherzustand nach Ausführen von $(\alpha_1\, ;\, \alpha_2)$ ist der, den man erreicht, wenn man zuerst α_1 ausführt und in dem dadurch erreichten Zustand α_2 ausführt,

(i) $[\![(\mathit{if}\ \tau\ \mathit{then}\ \alpha_1\ \mathit{else}\ \alpha_2)]\!]_h := \begin{cases} [\![\alpha_1]\!]_h, & \text{falls } [\![\tau]\!]_h = 1, \\ [\![\alpha_2]\!]_h, & \text{sonst} \end{cases},$

d.h. falls der Test τ erfüllt ist, ist es der durch Ausführen von α_1 erreichte Zustand, sonst der durch Ausführen von α_2 erreichte,

(j) $[\![(\mathit{while}\ \tau\ \mathit{do}\ \alpha)]\!]_h := \begin{cases} h, & \text{falls } [\![\tau]\!]_h = 0, \\ [\![(\mathit{while}\ \tau\ \mathit{do}\ \alpha)]\!]_{[\![\alpha]\!]_h}, & \text{sonst} \end{cases},$

d.h. falls der Test nicht erfüllt ist, ist der erreichte Zustand der unveränderte Zustand h, sonst ist es der Zustand, den man erreicht, wenn man zuerst α ausführt und in dem dadurch erreichten Zustand die *while*-Anweisung nochmal ausführt.

[18] Da dieser Zustand davon abhängt, was der zu Beginn der Ausführung vorliegende Zustand war, wäre es besser, als Bedeutung von α die durch das Ausführen von α bewirkte *Zustandsveränderung* $[\![\alpha]\!] : S \to S$ zu verstehen, d.h. $h \mapsto [\![\alpha]\!]_h$.

Beachte, daß die letzte Bedingung den Wert von $(\textit{while}\ \tau\ \textit{do}\ \alpha)$ bei der Belegung h nicht auf Werte kleinerer Ausdrücke zurückführt, sondern nur auf den Wert derselben Anweisung bei einer (möglicherweise anderen) Belegung $[\![\alpha]\!]_h$. Es ist also nicht gesichert, daß man überhaupt einen Wert definiert.

Beispielsweise terminiert die Auswertung der Anweisung

$$(\textit{while}\ 0 < 1\ \textit{do}\ x \leftarrow x)$$

nicht, sodaß $[\![(\textit{while}\ 0 < 1\ \textit{do}\ x \leftarrow x)]\!]_h(y)$ für jede Adresse y undefiniert ist.

3.3 Exkurs: Programmierung der Interpretation von NL

Die angegebene Interpretation der Beispielsprache NL verwendet Wahrheitsfunktionen, Testfunktionen und sogar Auswahlfunktionen, die anderen Funktionen einen Wert zuordnen.

Wir wollen nun demonstrieren, daß das keine „mathematische Spielerei“ ist, sondern tatsächlich programmiert werden kann. Weil in der angegebenen Semantik die Funktionen eine wichtige Rolle spielen, wählen wir eine *funktionale* Programmiersprache, nämlich *Standard ML*, kurz *SML*.

3.3.1 Die Interpretation $\mathcal{D}$ für NL

Zunächst geben wir die *Signatur* der Beispielsprache an, d.h. legen fest, wie die Kategorien und Konstanten dieser Kategorien heißen. (Der einzige Unterschied zu Beispiel 3.4 ist, daß wir die Kategorie der Verben in intransitive und transitive Verben und die der Konjunktionen in ein- und zweistellige Konjunktionen aufspalten, weil die Programmiersprachen eine feste Stelligkeit erwarten.)

$\langle$*NL/nl.sml*$\rangle\equiv$

```
signature INTERPRETATION =
  sig
    datatype person = a | b | c | d | e | f | g | h | i | j | k
                    | l | m | n | p | q | r | s | t | u | v | w

    type PN
    type NP
    type S = bool
    type N
    type DET
    type COP
    type TV
    type IV
    type CONJ1
    type CONJ2

    val Maria : PN
    val Emil : PN
    val Anna : PN
    val Frau : N
    val Mann : N
    val Student : N
    val arbeitet : IV
    val kennt : TV
    val liebt : TV
```

```
    val ist : COP
    val der : DET
    val die : DET

    val nicht : CONJ1
    val und : CONJ2
    val oder : CONJ2

    val toString : person -> string
  end
```

Jede Interpretation muß dann für diese Kategorien und diese Konstanten bestimmte Wertebereiche und Werte angeben. Unsere Beispielinterpretation muß also entsprechende Definitionen für die Typen (Kategorien) und die Bedeutungen der Konstanten angeben. Das sieht so aus:

⟨*NL/nl.sml*⟩+≡

```
 structure D : INTERPRETATION =
  struct
    (* Wertebereiche der syntaktischen Kategorien: *)

    datatype person = a | b | c | d | e | f | g | h | i | j | k
                    | l | m | n | p | q | r | s | t | u | v | w

    type PN = person
    type NP = person
    type S = bool
    type N = person -> bool
    type DET = (person -> bool) -> person
    type COP = person * person -> bool
    type IV = (person -> bool)
    type TV = (person * person -> bool)
    type CONJ1 = (bool -> bool)
    type CONJ2 = (bool * bool -> bool)

    (* Werte der einfachen Ausdruecke: *)

    val Anna = a
    val Emil = e
    val Maria = m

    fun nicht(x) = not x
    fun und(x,y) = x andalso y
    fun oder(x,y) = x orelse y

    fun Frau( a | b | c | g | h | i | m | n | s | t | u ) = true
      | Frau(_) = false
```

```
    fun Mann( d | e | f | j | k | l | p | q | r | v | w ) = true
      | Mann(_) = false

    fun Student( u | v | w ) = false
      | Student(_) = true

    fun arbeitet( a | b | c | d | e ) = true
      | arbeitet(_) = false

    fun kennt( (a,e) | (a,m) | (e,a) | (m,e) ) = true
      | kennt(x,y) = if x=y then true else false

    fun liebt( (e,m) | (a,e) ) = true
      | liebt(_) = false

    fun der(F) = case (List.find F [a,b,c,d,e,f,g,h,i,j,k,
                                    l,m,n,p,q,r,s,t,u,v,w])
                   of SOME x => x | NONE => a
    fun die(F) = der(F)

    fun ist(x,y) = if x=y then true else false

    val toString = (fn a => "a" | b => "b" | c => "c"
                      | d => "d" | e => "e" | m => "m"
                        | _ => "some person")
  end
```

Mit `List.find F liste` wird die Funktion `F` der Reihe nach auf die Elemente der Liste `liste` angewendet, bis ein Element `x` gefunden wird, wo `F(x) = true` ist; dann wird `SOME x` ausgegeben, und falls kein solches `x` gefunden wird, wird `NONE` ausgegeben. Unsere Interpretation für `der` gibt im ersten Fall `x` und im zweiten Fall `a` aus.

Die Hilfsfunktion `toString` dient nur dazu, später die Personenwerte ausdrucken zu können.

3.3.2 Grammatik und Auswertung für NL

Um die Auswertungsfunktion auch testen zu können, wollen wir Ausdrücke unserer Sprache NL als Zeichenreihen am Terminal eingeben können und den Wert berechnen lassen. Dazu brauchen wir einen *Parser* – ein Programm, was die Syntaxanalyse der Eingabe besorgt, also feststellt, ob die Zeichenreihe ein erlaubter Ausdruck von NL ist und wie dieser aufgebaut ist.

Ein solcher Parser kann mit einem *Parsergenerator* aus einer Grammatik der Sprache erzeugt werden (wenn die Grammatik nicht zu kompliziert ist). Dabei kann man dem Parsergenerator auch sagen, wie er nach der syntaktischen Analyse die Ausdrücke α der verschiedenen Kategorien τ auswerten soll.

Ein solcher Parsergenerator ist *ML-Yacc* [7]. Man definiert die gewünschte Sprache in einem bestimmten Format für kontextfreie Grammatiken. Bei den einzelnen Kategorien gibt man den gewünschten Wertebereich an, z.B. für `satz` und `np` die in der Interpretation `D` vorgesehenen Bereiche `D.S` und `D.NP`:

⟨*Syntaktische Kategorien mit Typ der Werte*⟩≡
```
%nonterm
   start of (value * string) option
    | satz of D.S
    | np of D.NP
```

(Als `start`-Symbol werden `satz` und `np` erlaubt, und deren mögliche Werte fassen wir unten in einen Bereich `value` zusammen; dazu ergänzen wir noch eine `string`-Komponente, die zum Ausdrucken des Werts benutzt werden soll.)

Bei den Grammatikregeln zu `satz, np, ..` usw. steht links ein syntaktisches Muster α und rechts die jeweilige Auswertungsbedingung für $[\![\alpha]\!]^{\mathcal{D}}$, z.B. besagt

⟨*Syntaxregel mit Auswertungsbedingung*⟩≡
```
satz : np IST np              (D.ist(np1,np2))
```

daß der Wert eines Satzes der Form `np IST np` mit der in `D` festgelegten Bedeutung `D.ist` von `ist` aus den Werten `np1` der ersten und `np2` der zweiten Nominalphrase berechnet werden soll.

Man vergleiche die unten angegebenen Auswertungsbedingungen zu den einzelnen Regeln mit denen aus Beipiel 3.4.

⟨*NL/nl.grm*⟩≡
```
datatype value = object of D.PN | truth of bool
exception Undefined

%%
%header (functor NL_LrValsFun (structure Token : TOKEN) : NL__LRVALS)

%term
     EOF | PUNKT
   | UND | ODER | NICHT
   | DET
   | IST
   | pn of D.PN
   | n of D.N
   | iv of D.IV
   | tv of D.TV

%nonterm
   start of (value * string) option
```

```
    | satz of D.S
    | np of D.NP

%start start

%eop EOF PUNKT
%noshift EOF PUNKT
%pos int

%left UND
%left ODER

%name NL_
%verbose

%%

start : satz            (SOME (truth(satz),Bool.toString satz))
      | np              (SOME (object(np), D.toString np))
      |                 (NONE)

satz : np IST np        (D.ist(np1,np2))
     | np IST NICHT np  (D.nicht(D.ist(np1,np2)))
     | np iv            (iv(np))
     | np iv NICHT      (D.nicht(iv(np)))
     | np tv np         (tv(np1,np2))
     | np tv np NICHT   (D.nicht(tv(np1,np2)))
     | satz UND satz    (D.und(satz1,satz2))
     | satz ODER satz   (D.oder(satz1,satz2))

np : pn                 (pn)
   | DET n              (D.der n)
```

Der Parser erwartet, daß die lexikalische Analyse, die Erkennung der erlaubten Wörter, getan ist und ihm passende *token* liefert, d.h. Ausdrücke der Terminalkategorien oder deren Bedeutung. Wir definieren die lexikalische Analyse im folgenden Abschnitt so, daß sie für Personennamen, Verben und Gemeinnamen als Token die jeweiligen Werte laut der Interpretation lieft. (Dann kann man die Beispielsprache um Wörter dieser Kategorien erweitern, indem man nur die Interpretation und die lexikalische Analyse anpaßt.) Dadurch wird allerdings die Auswertungsfunktion teilweise in `nl.lex` und teilweise in `nl.grm` angegeben.

Bei Wörtern anderer Kategorien wie den Junktoren oder der Kopula ist es klüger, die Bedeutung nicht im Lexer festzulegen, da sie vielleicht vom syntaktischen Kontext abhängt: man könnte etwa eine Konjunktion von Sätzen und eine Konjunktion von Nominalphrasen erlauben und braucht dann für „und“ verschiedene Bedeutungen, unter denen man je nach dem syntaktischen Kontext wählen muß.

3.3.3 Lexikalische Analyse für NL

Ein Programm zur lexikalischen Analyse, ein *Lexer*, kann ebenfalls automatisch aus einer Beschreibung der erlaubten Token erfolgen, mit einem *Lexergenerator*.

Die nach außen sichtbaren Teile des Lexers und Parsers werden in einer Schnittstelle festgelegt, die Zeilennummern und Postionen in der Eingabe verwaltet, mit der lexikalische oder syntaktische Fehler mit der Fehlerstelle ausgegeben werden können. (Wir geben aber keine vernünftige Fehlerbehandlung an.)

⟨*NL/nl.interface.sml*⟩≡

```
signature INTERFACE =
  sig
      type pos
      val line : pos ref
      val init_line : unit -> unit
      val next_line : unit -> unit
      val error : string * pos * pos -> unit

      type arg
      val nothing : arg
  end

functor Interface () : INTERFACE =
  struct
    type pos = int
    val line = ref 0
    fun init_line () = (line := 1)
    fun next_line () = (line := !line + 1)
    fun error (errmsg,line:pos,_) =
        print("Line " ^ (Int.toString line) ^ ": " ^ errmsg ^ "\n")

    type arg = unit
    val nothing = ()
  end
```

Zur Erzeugung des Lexers für NL benutzen wir den Lexergenerator *MLLex*. Er erwartet eine Beschreibung der zu erkennenden Wörter in einem bestimmten Format. Für die Wörter von NL sieht die Beschreibung so aus:

⟨*NL/nl.lex*⟩≡

```
structure Tokens = Tokens
structure Interface = Interface
```

```
open Interface

type pos = Interface.pos
type svalue = Tokens.svalue
type ('a,'b) token = ('a,'b) Tokens.token
type lexresult = (svalue,pos) token

fun eof() = Tokens.EOF(!line,!line)
fun makeInt (s : string) = s

%%
%header (functor NL_LexFun(structure Tokens: NL__TOKENS
                           structure Interface: INTERFACE) : LEXER);
%full

idchars=[0-9]|[_&+/@|#*`'><];
alpha=[a-zA-ZÄÖÜäöüß];
comment=%(.*)\n;
whitespace=[\t\ ]*;
%%
<INITIAL>{whitespace} => (lex());
<INITIAL>{comment}    => (next_line(); lex());
<INITIAL>\n           => (next_line(); lex());
<INITIAL>"Der"        => (Tokens.DET(!line,!line));
<INITIAL>"Die"        => (Tokens.DET(!line,!line));
"."         => (Tokens.PUNKT(!line,!line));
"und"       => (Tokens.UND(!line,!line));
"oder"      => (Tokens.ODER(!line,!line));
"nicht"     => (Tokens.NICHT(!line,!line));
"der"       => (Tokens.DET(!line,!line));
"die"       => (Tokens.DET(!line,!line));
"ist"       => (Tokens.IST(!line,!line));
"Anna"      => (Tokens.pn(D.Anna,!line,!line));
"Emil"      => (Tokens.pn(D.Emil,!line,!line));
"Maria"     => (Tokens.pn(D.Maria,!line,!line));
"Mann"      => (Tokens.n(D.Mann,!line,!line));
"Frau"      => (Tokens.n(D.Frau,!line,!line));
"Student"   => (Tokens.n(D.Student,!line,!line));
"arbeitet"  => (Tokens.iv(D.arbeitet,!line,!line));
"kennt"     => (Tokens.tv(D.kennt,!line,!line));
"liebt"     => (Tokens.tv(D.liebt,!line,!line));
.           => (error ("ignoring illegal character " ^ yytext,
                       !line,!line); lex());
```

Im wesentlichen erkennen wir also die einzelnen Zeichenreihen wie `"und"` oder `"Anna"` und geben sie oder ihren Wert als Token aus.

3.3.4 Linken: Verbinden von Lexer und Parser für NL

Man muß nun mit `mllex` aus der Datei `nl.lex` einen Lexer und mit `mlyacc` aus der Datei `nl.grm` einen Parser erzeugen; wir automatisieren das unten. Wenn das getan ist, muß man diese Programme zusammenbinden. Dadurch wird ein *SML*-Paket `NL` erzeugt, in dem die Funktion `parse` zur Analyse (und Auswertung!) von *NL*-Ausdrücken enthalten ist.

Das Verbinden geschieht mit folgendem Programm:

⟨*NL/nl.link.sml*⟩≡

```
structure Interface : INTERFACE = Interface();

structure NL_LrVals : NL__LRVALS =
   NL_LrValsFun(structure Token = LrParser.Token);

structure NL_Lex : LEXER =
   NL_LexFun(structure Tokens = NL_LrVals.Tokens
             structure Interface = Interface);

structure NL_Parser : PARSER =
   Join(structure ParserData = NL_LrVals.ParserData
        structure Lex = NL_Lex
        structure LrParser = LrParser);

signature PARSE =
  sig
    val parse : unit -> (NL_Parser.svalue,NL_Parser.pos) LrParser.Token.token
    exception Undefined
  end

functor Parse (structure Interface : INTERFACE
               structure Parser : PARSER
                  sharing type Parser.arg = Interface.arg
                  sharing type Parser.pos = Interface.pos
               structure Tokens : NL__TOKENS
                  sharing type Tokens.token = Parser.Token.token
                  sharing type Tokens.svalue = Parser.svalue
               ) : PARSE =
  struct
    exception Undefined

    fun invoke lexstream =
        let val print_error =
                fn (s,i:int,_) =>
                   TextIO.output(TextIO.stdOut,"Error, line "
                                                ^ (Int.toString i)
```

```
                                                    ^ ", " ^ s ^ "\n")
          in
            NL_Parser.parse(0,lexstream,print_error,())
          end

      fun parse () =
          let
            fun reader (n:int) =
                case (TextIO.inputLine TextIO.stdIn) of
                  SOME s => s
                | NONE => raise Undefined

            (* Für SML 110.0.3 stattdessen:
               fun reader (n:int) = TextIO.inputLine TextIO.stdIn *)

            val dummyEOF = NL_LrVals.Tokens.EOF(0,0)

            fun loop lexer =
                let val (result,lexer) = invoke lexer
                    val (nextToken,lexer) = NL_Parser.Stream.get lexer
                    val _ = case result of
                              (SOME (_,s)) =>
                                 TextIO.output(TextIO.stdOut,
                                               "Value is: " ^ s ^ "\n")
                             | NONE =>
                                 TextIO.output(TextIO.stdOut,
                                  "Value is neither a Boolean nor an object.\n")
                in if NL_Parser.sameToken(nextToken,dummyEOF)
                   then nextToken
                   else loop lexer
                end
          in
            loop (NL_Parser.makeLexer reader)
          end
    end

  structure NL : PARSE =
     Parse (structure Interface = Interface
            structure Parser = NL_Parser
            structure Tokens = NL_LrVals.Tokens );
```

Für SML/NJ Version 110.0.3 muß man die Funktion `reader` wie im Kommentar angegeben ändern; die so geänderte Datei heiße `nl.link.110.0.3.sml`.

3.3.5 Erstellung des Programmpakets für NL

Die Sprache *SML*, genauer gesagt die Implementierung *SML of New Jersey*, hat einen sogenannten *Compilation Manager*, der –analog zu `make` unter Unix– das Übersetzen und Verbinden von Programmen nach einer passenden Beschreibung vornimmt.

Man gibt dem *Compilation Manager* in einer Beschreibungsdatei `*.cm` (analog zu einem `Makefile` unter Unix) an, daß gewisse Bibliotheksdateien von `smlnj` und `ml-yacc` geladen werden sollen. Außerdem müssen die Interpretation `nl.sml` und die Beschreibungen `nl.lex` und `nl.grm` zur Erzeugung des Lexers und Parsers geladen werden; das Interface `nl.interface.sml` wird in `nl.lex` benutzt und muß ebenfalls geladen werden. Schließlich wird auch noch die Beschreibung `nl.link.sml` zum Verbinden von Lexer und Parser gebraucht.

Die Beschreibungsdatei ist für Version 110.42 die folgende:

⟨*NL/nl.110.42.cm*⟩≡

```
Library
  structure NL
is
  $/smlnj-lib.cm
  $/ml-yacc-lib.cm
  $/basis.cm

  nl.sml        (* Interpretationen D und Q *)
  nl.interface.sml (* fuer Lexer und Linker *)
  nl.link.sml   (* an der LMU: nl.link.110.0.3.sml *)
  nl.lex        (* fuer Tokenizer und Parser *)
  nl.grm
```

Diese Dateien `nl.grm`, `nl.lex`, `nl.sml`, `nl.interface.sml`, `nl.link.sml` und `nl.110.42.cm` sind im Archiv `NL.tgz` (auf der Internetseite des Kurses) verpackt. Damit kann man `NL` wie folgt installieren:

(a) Man packt das Archiv mit `tar -xvfz NL.tgz` aus, geht mit `cd NL` in das entstehende Verzeichnis `NL`.

(b) Dort startet man SML/NJ Version 110.42 (oder eine spätere Version) mit `/usr/bin/sml` (bzw. `sml-110.0.42`, vgl. Bemerkung 3.12.)

(c) In SML fordert man den Compilation Manager auf, das Paket `NL` gemäß der Beschreibungsdatei zu erzeugen:

```
Standard ML of New Jersey v110.45 [FLINT v1.5], February 13, 2004
- CM.make "nl.110.42.cm";
```

Das sollte zu einer Reaktion wie der folgenden führen:

```
[autoloading]
[library $smlnj/cm/cm.cm is stable]
[library $smlnj/internal/cm-sig-lib.cm is stable]
[library $/pgraph.cm is stable]
[library $smlnj/internal/srcpath-lib.cm is stable]
[autoloading done]
[scanning nl.110.42.cm]
[library $/ml-yacc-lib.cm is stable]
[/usr/local/smlnj/bin/ml-lex nl.lex]

Number of states = 80
Number of distinct rows = 56
Approx. memory size of trans. table = 14336 bytes
[parsing (nl.110.42.cm):nl.lex.sml]
[/usr/local/smlnj/bin/ml-yacc nl.grm]
[parsing (nl.110.42.cm):nl.grm.sig]
[parsing (nl.110.42.cm):nl.grm.sml]
[loading (nl.110.42.cm):nl.interface.sml]
[loading (nl.110.42.cm):nl.sml]
[compiling (nl.110.42.cm):nl.grm.sig]
[code: 52, env: 678 bytes]
[compiling (nl.110.42.cm):nl.grm.sml]
[code: 16637, data: 829, env: 1420 bytes]
[compiling (nl.110.42.cm):nl.lex.sml]
[code: 32498, data: 16350, env: 934 bytes]
[loading (nl.110.42.cm):nl.link.sml]
[New bindings added.]
val it = true : bool
```

Die Erzeugung des Lexers und Parsers war erfolgreich; unter den `new bindings` ist die Struktur `NL`, die die Analyse- und Auswertungsfunktion `parse` enthält. (Die erzeugten Lexer und Parser werden unter `NL/nl.lex.sml` und `NL/nl.grm.sml` gespeichert. Bei erneuter Ausführung von `CM.make` werden nur noch die Teile übersetzt, deren Quelldateien sich geändert haben.)

Bemerkung 3.12 Auf den Rechnern der Informatik an der LMU ist `SML/NJ version 110.42` durch `sml-110.0.42` zu starten. Man muß in der Datei `nl.cm` die Zeile `nl.link.sml` durch `nl.link.110.0.3.sml` ersetzen. (In dieser Version des Linkers ist die im Kommentar oben angegebene Änderung der Funktion `reader` vorgenommen.)

Siehe auch die Kurzdokumentation `NL/README` auf Seite 52.

3.3.6 Anwenden des Parsers und Auswertung von NL-Ausdrücken

Nun kann man den Parser aufrufen und Beispiele testen:

```
- NL.parse();
Anna kennt Emil.
Value is: true
Emil ist nicht der Student.
Value is: true
Anna ist der Student.
Value is: true
Anna ist Emil.
Value is: false
Emil kennt Anna und Emil liebt Anna nicht.
Value is: true
der Mann kennt die Frau nicht.
Value is: true
der Mann ist Emil.
Value is: false
```

Ebenso kann man den Wert von Nominalphrasen ausrechnen und anzeigen lassen:

```
- NL.parse();
Emil.
Value is: e
der Mann.
Value is: d
Anna.
Value is: a
die Frau.
Value is: a
der Student.
Value is: a
```

Da die Grammatik keine Konjunktion von Nominalphrasen kennt, werden diese nicht erkannt:

```
NL.parse();
der Student und die Frau.
Error, line 1, syntax error found at UND

uncaught exception ParseError
raised at: ../ml-yacc/lib/parser2.sml:514.28-514.38
-
```

Die Konjunktion von Nominalphrasen ist in NL nicht nur syntaktisch inkorrekt. Selbst wenn eine solche Konjunktion als Nominalphrase zugelassen wird, kann der Wert von `der Student und die Frau` ja wohl kein Individuum und daher nicht in $D_{NP} = D_{PN}$ sein – er ist vielleicht ein *Paar von Individuen*, und dafür braucht man einen anderen Wertebereich $D_{NP} \neq D_{PN}$.

Man muß in `nl.grm` selbst dafür sorgen, daß die Werte, die in lesbarer Form ausgegeben werden sollen, erst in eine Zeichenreihe übersetzt werden. Wir haben das nur für die Kategorien `satz` und `np` vorgesehen. Für die übrigen Kategorien `n`, `det` usw. hätte es keinen Sinn, da die Werte ja Funktionen sind und die Funktionsdefinitionen zu Maschinenprogrammen übersetzt werden, die wir sowieso nicht lesen könnten.

Problem 3.13 Man kann zwar Funktionale `F` wie in

```
fun F(f) = (... f(x) ... f(y) ...)
```

definieren, die die Argumentfunktion `f` anwenden, aber z.B. keine, die eine Fallunterscheidung nach Eigenschaften von `f` machen, wie in

$\langle$*nicht legal*$\rangle\equiv$

```
fun Id(f:person-> person) =
    if (f = (fn x => x)) then true else false
```

Das gilt auch für endliche Funktionen und hat nichts mit Eigenheiten von *SML* zu tun. Wie weit kommt man also mit einer funktionalen Interpretation von Relationen?

4 Quantoren und Variable

Die bisherigen Sprachfragmente erlauben noch keine Auswertung von *quantifizierten Ausdrücken* wie „einige Studenten“, „die meisten Frauen“, „ein Drittel der Männer“ oder „alle natürliche Zahlen“.

Die klassische Logik hat solche Ausdrücke (im heute üblichen Sinne von Nominalphrasen) gar nicht als Ausdrücke aufgefaßt und ihnen höchstens indirekt eine Bedeutung zugeordnet. Dennoch wurden zumindest die einzelnen Wörter „einige“, „alle“ usw. als Ausdrücke anerkannt, aber auch für diese wurden keine Bedeutungen oder Werte angegeben.

Definition 4.1 Ein Ausdruck α hat eine *kategorematische Bedeutung* oder ist ein *Kategorem*, wenn er eine syntaktische Kategorie σ und einen Wert $[\![\alpha]\!]$ in dem Wertebereich (der „semantischen Kategorie“) D_σ hat. Andernfalls hat α nur eine *synkategorematische Bedeutung* oder ist ein *Synkategorem.*

Die synkategorematischen Ausdrücke sind nicht bedeutungs*los*; nur wird ihre Bedeutung nicht als Wert einer der Bereiche D_σ angegeben, sondern implizit in der Definition der Auswertungsfunktion $[\![\cdot]\!]^{\mathcal{D}}$. [19]

Beispiel 4.2 In der klassischen Logik wird beispielsweise durch

$$a^{\mathcal{D}}(A,B) := \begin{cases} 1, & \text{falls } A \subseteq B, \\ 0 & \text{sonst} \end{cases}$$

die Bedeutung von „alle“ aus

$$[\![\text{alle Menschen sind Lebewesen}]\!] := a^{\mathcal{D}}([\![\text{Mensch}]\!], [\![\text{Lebewesen}]\!])$$

in die Bedeutung der Kopula a eingebaut: $A \subseteq B$ enthält ja den „All“quator $\forall$, wenn man es durch $\forall a(a \in A \to a \in B)$ ausdrückt. *Nominalphrasen* wie „alle Menschen“ gibt es nicht, folglich auch keine Werte $[\![\text{alle Menschen}]\!]$.

4.1 Die Quantorenlogik QL mit einem Objektbereich

Wir wollen nun das Beispiel der Prädikatenlogik zur Quantorenlogik ausweiten und dabei den Quantoren eine kategorematische Bedeutung geben.

Beispiel 4.3 Die Sprache QL der Quantorenlogik hat neben den Kategorien der Prädikatenlogik zwei weitere syntaktische Kategorien: die der Quantoren

[19] Damit auch weiterhin jedem *Ausdruck* α einer syntaktischen Kategorie σ immer ein Wert $[\![\alpha]\!] \in D_\sigma$ entspricht, sollte man bei den Synkategoremata eigentlich gar nicht von Ausdrücken einer syntaktischen Kategorie sprechen. Das tut man oft auch nicht, z.B. wenn man die Funktionswörter „und“ usw. erlaubt, ohne die syntaktische Kategorie der Junktoren in die Sprachbeschreibung einzuführen.

und die der (Individuen-) Variable:

Synt. Kategorie		Metavariable und Konstruktionsmöglichkeiten	
Funktor	F	$f ::=$	$f_1 \mid f_2 \mid \ldots$
Prädikat	P	$\rho ::=$	$P_1 \mid P_2 \mid P_3 \mid \cdots$
Junktor	J	$o ::=$	$\neg \mid \wedge \mid \vee \mid \rightarrow \mid \leftrightarrow$
Variable	V	$x ::=$	$x_1 \mid x_2 \mid \ldots$
Term	T	$t ::=$	$x \mid f^{(n)}(t_1, \ldots, t_n)$
Quantor	Q	$Q ::=$	$\forall \mid \exists$
Formel	Fml	$\varphi ::=$	$\rho^{(n)}(t_1, \ldots, t_n) \mid \neg\varphi \mid (\varphi_1\ o^{(2)}\ \varphi_2) \mid Qx\varphi$

(Die Stelligkeit der Prädikate und Funktoren lassen wir wieder implizit.)

Beachte, daß auch die Quantorenlogik keine Nominalphrasen im üblichen Sinn kennt: am ehesten entsprechen den Nominalphrasen die Terme, aber *es gibt in QL keine quantifizierten Terme*, sondern nur *quantifizierte Formeln*, und natürlichsprachliche Ausdrücke wie *alle Menschen* bezeichnen weder Individuen noch Wahrheitswerte.

Wir ändern die Interpretation der prädikatenlogischen Sprache PL (vgl. Beispiel 3.2) wie folgt:

(a) Werte der Variablen sind die *Individuen* des Diskursuniversums:

$$D_V := U \quad (= D_T) \tag{19}$$

(b) Werte der Quantoren sind *Testfunktionen auf Testfunktionen auf Individuen*, d.h.

$$D_Q := (D_T \rightarrow \mathbb{B}) \rightarrow \mathbb{B}, \tag{20}$$

Darunter der Existenztest, der durch

$$\exists^{\mathcal{D}}(f) := \begin{cases} 1, & \text{falls } f(a) = 1 \text{ für mindestens ein } a \in D_T \text{ ist,} \\ 0, & \text{sonst,} \end{cases}$$

und der Alltest, der durch

$$\forall^{\mathcal{D}}(f) := \begin{cases} 1, & \text{falls } f(a) = 1 \text{ für alle } a \in D_T \text{ ist,} \\ 0, & \text{sonst,} \end{cases}$$

definiert ist. ($\exists^{\mathcal{D}}$ testet, ob der Test $f : D_T \rightarrow \mathbb{B}$ für mindestens ein Individuum positiv ausgeht, und $\forall^{\mathcal{D}}$ testet, ob f für alle Individuen positiv ausgeht.)

Eine Interpretation $\mathcal{D}$ soll für die einfachen Ausdrücken *bestimmte* Bedeutungen festlegen, aber Variable sollen keine bestimmten, sondern „variable“ oder „variierende“ Bedeutung haben. Daher wird der Wert von Variablen und allen

Ausdrücken, die Variable enthalten, nur relativ zu einer „momentanen" Bedeutung der Variablen definiert, d.h. einer *Belegung* $h : \mathit{Var} \to U$, die einer Variablen x ihre momentane Bedeutung $h(x)$ aus dem Diskursuniversum $U = D_T$ der Objekte, über die bei der Interpretation $\mathcal{D}$ gesprochen wird, zuordnen. Die Auswertungsfunktion $[\![\cdot]\!]$ braucht also stets ein weiteres Argument: statt $[\![\alpha]\!]^{\mathcal{D}}$ kann man nur $[\![\alpha]\!]^{\mathcal{D}}(h)$ für Belegungen h definieren.

Bei den Konstanten spielt die Belegung natürlich keine Rolle:

(a) $[\![f]\!](h) := f^{\mathcal{D}}$ für Funktoren f,

(b) $[\![\rho]\!](h) := \rho^{\mathcal{D}}$ für Prädikate ρ

(c) $[\![o]\!](h) := o^{\mathbb{B}}$ für Junktoren o,

(d) $[\![Q]\!](h) := Q^{\mathcal{D}}$ für Quantoren Q,

Bei den Termen und Formeln spielt die Belegung für die Auswertung eine Rolle, da diese im Allgemeinen Variable enthalten:

(e) $[\![x]\!](h) := h(x)$,

(f) $[\![f(t_1, \ldots, t_n)]\!](h) := f^{\mathcal{D}}([\![t_1]\!](h), \ldots, [\![t_n]\!](h))$,

(g) $[\![\rho(t_1, \ldots, t_n)]\!](h) := \rho^{\mathcal{D}}([\![t_1]\!](h), \ldots, [\![t_n]\!](h))$,

(h) $[\![\neg\varphi]\!](h) := \neg^{\mathcal{D}}([\![\varphi]\!](h))$,

(i) $[\![(\varphi_1 \; o \; \varphi_2)]\!](h) := o^{\mathbb{B}}([\![\varphi_1]\!](h), [\![\varphi_2]\!](h))$,

(In den letzten vier Fällen haben wir den Umweg über (a)-(c) abgekürzt, was auch in Zukunft oft gemacht werden wird.) Für den verbleibenden Fall von Formeln der Form $\forall x\varphi$ und $\exists x\varphi$ soll der Wert von x variieren, d.h. mit x ist nicht das von h abhängende Objekt $[\![x]\!](h) = h(x)$ gemeint. Für jedes $a \in D_T$ definieren wir eine Belegung $h[x/a]$ durch

$$h[x/a](y) := \begin{cases} h(y), & \text{falls } y \neq x, \\ a, & \text{sonst,} \end{cases}$$

die sich von h höchstens bei der Variablen x unterscheidet, wo sie a liefert. Damit können wir die Auswertung für die quantifizierten Formeln festlegen:

(j) $[\![Qx\varphi]\!](h) := Q^{\mathcal{D}}(\chi_{\{a \in D_T \,|\, [\![\varphi]\!](h[x/a])=1\}})$.

Verfolgt man die obigen Definitionen, so besagen sie:

$$\begin{aligned} [\![\forall x\varphi]\!](h) &= \begin{cases} 1, & \text{falls } [\![\varphi]\!](h[x/a]) = 1 \text{ für jedes } a \in D_T, \\ 0, & \text{sonst.} \end{cases} \\ [\![\exists x\varphi]\!](h) &= \begin{cases} 1, & \text{falls } [\![\varphi]\!](h[x/a]) = 1 \text{ für mindestens ein } a \in D_T, \\ 0, & \text{sonst.} \end{cases} \end{aligned}$$

Die Belegung h trägt zur Bedeutung von $Qx\varphi$ nur die Werte $h(y)$ der *nicht* durch Quantoren „gebundenen" Variablen bei, insbesondere spielt also $h(x)$ für $[\![Qx\varphi]\!](h)$ keine Rolle:

Beispiel 4.4 Sei $\mathcal{D}$ eine Interpretation mit $b \in D_T$ und $\doteq$ ein Prädikat der Sprache, dessen Bedeutung $\doteq^{\mathcal{D}}$ die übliche Gleichheit auf D_T sei. Für eine Belegung $h : Var \to D_T$ mit $h(x_1) = b = h(x_2)$ ist

$$\begin{array}{ll} [\![\forall x_2(x_1 \doteq x_2)]\!](h) = 1 & \\ \iff & \text{für jedes } a \in D_T \text{ ist } [\![(x_1 \doteq x_2)]\!](h[x_2/a]) = 1 \\ \iff & \text{für jedes } a \in D_T \text{ ist } [\![x_1]\!](h[x_2/a]) = [\![x_2]\!](h[x_2/a]) \\ \iff & \text{für jedes } a \in D_T \text{ ist } h[x_2/a](x_1) = h[x_2/a](x_2) \\ \iff & \text{für jedes } a \in D_T \text{ ist } h(x_1) = a \\ \iff & \text{für jedes } a \in D_T \text{ ist } b = a. \end{array}$$

Für die Bedeutung spielt also zwar der Wert $[\![x_1]\!]h = [\![x_1]\!](h[x_2/a])$ eine Rolle, aber nicht $[\![x_2]\!](h)$, da stattdessen $[\![x_2]\!](h[x_2/a]) = a$ benutzt wird, was für $a \neq b$ von $[\![x_1]\!](h) = b$ verschieden ist.

Beachte, daß bei der Berechnung von $[\![\forall x\varphi \wedge \psi]\!](h) = \wedge^{\mathbb{B}}([\![\forall x\varphi]\!](h), [\![\psi]\!](h))$ der Wert $h(x)$ zwar keine Rolle für $[\![\forall x\varphi]\!](h)$ spielt, aber sehr wohl für $[\![\psi]\!](h)$. Der *Wirkungsbereich* des Quantors Q in $Qx\varphi$ ist die direkt folgende Formel φ (genauer gesagt: abzüglich der Wirkungsbereiche der Teilformeln $Q'x'\varphi'$ mit $x' = x$).

Bemerkung 4.5 Wir haben (einstellige) Prädikate als Tests auf Individuen, also als Funktionen $f : D_T \to \mathbb{B}$, und (einstellige) Quantoren als Tests auf solchen Prädikaten, also als Funktionen $Q : (D_T \to \mathbb{B}) \to \mathbb{B}$, interpretiert. Wenn man statt der Tests $\chi : A \to \mathbb{B}$ die Urbildmengen $R_\chi := \{a \in A \mid \chi(a) = 1\}$ nimmt, entsprechen den Prädikaten Mengen $R_f \subseteq D_T$ von Individuen und den Quantoren Mengen $R_Q \subseteq 2^{D_T}$ von (den Prädikaten entsprechenden) Mengen, sodaß

$$Q(f) = 1 \iff R_f \in R_Q.$$

Insbesondere ist

$$R_\exists = \{R \subseteq D_T \mid R \neq \emptyset\} \quad \text{und} \quad R_\forall = \{R \subseteq D_T \mid R = D_T\}.$$

Quantoren sind also Prädikate von Prädikaten von Individuen. (Analog für mehrstellige Prädikate.)

Beispiel 4.6 Als Beispiel für einen zweistelligen Quantor betrachten wir den Quantor in

Mehr Elemente von A als von B erfüllen φ, was durch $Mx(A(x) \wedge \varphi, B(x) \wedge \varphi)$ auf den Quantor M für *mehr-als* zurückgeführt wird. Die Bedeutung von M in einer Interpretation $\mathcal{D}$ mit Diskursuniversum $U = D_{Var}$ ist

$$M^{\mathcal{D}}(\chi_1, \chi_2) \quad := \quad \begin{cases} 1, & \text{falls } |\{a \in U \mid \chi_1(a) = 1\}| > |\{a \in U \mid \chi_2(a) = 1\}| \\ 0, & \text{sonst,} \end{cases}$$

d.h. $M^{\mathcal{D}}(\chi_1, \chi_2)$ ist wahr, wenn *mehr* Objekte aus U den Test χ_1 bestehen *als* den Test χ_2. Damit definieren wir den Wert von neuen Formeln durch

$$[\![Mx(\varphi_1, \varphi_2)]\!](h) := \begin{cases} 1, & \text{falls } |\{a \mid [\![\varphi_1]\!](h[x/a]) = 1\}| > |\{a \mid [\![\varphi_2]\!](h[x/a]) = 1\}| \\ 0 & \text{sonst.} \end{cases}$$

Der Quantor M kommt in Sätzen wie *Mehr Männer als Frauen lieben Fußball* in der natürlichen Sprache vor.

4.2 Quantifizierung über verschiedene Bereiche

Um die Bedeutung der quantifizierten Formeln $\exists x\varphi$ und $\forall x\varphi$ zu erklären, haben wir Variable x und Belegungen $h : Var \to D_{Var}$ gebraucht. In der Quantorenlogik beschränkt man sich normalerweise auf *eine* Art von Variablen und *einen* Bereich D_{Var} von Objekten, „über den die Variablen laufen“, z.B.

(a) in der Mengenlehre das Mengenuniversum $\mathbb{V}$ für $\mathcal{D} = (\mathbb{V}, \epsilon)$,

(b) in der Zahlentheorie den Zahlbereich $\mathbb{Z}$ für $\mathcal{D} = (\mathbb{Z}, +, \cdot, 0, 1)$.

Wenn man über *verschiedene* Arten von Objekten quantifizieren will, z.B. über Zahlen und Mengen, braucht man für verschiedene Kategorien σ auch verschiedene Variablenmengen Var_σ, Wertebereiche $D_{Var_\sigma} := D_\sigma$, Belegungen $h_\sigma : Var_\sigma \to D_\sigma$ und Quantoren Q_σ, und hat Interpretationen

$$\mathcal{D} = (D_\sigma, Q_\sigma^{\mathcal{D}}, c_\sigma^{\mathcal{D}}, \ldots)$$

mit $Q_\sigma^{\mathcal{D}} : (D_\sigma \to \mathbb{B}) \to \mathbb{B}$ für die Quantoren und $c_\sigma^{\mathcal{D}} \in D_\sigma$ für die Konstanten der Kategorie σ, z.B.

$$\tilde{\mathcal{D}} = (\mathbb{V}, \exists_{\mathbb{V}}, \forall_{\mathbb{V}}, \epsilon, \mathbb{Z}, \exists_{\mathbb{Z}}, \forall_{\mathbb{Z}}, +, \cdot, 0, 1),$$

wenn man über Mengen und Zahlen quantifizieren will. Variable müssen durch die Kategorie σ der Objekte, über die sie laufen, gekennzeichnet werden, z.B. x_σ statt x. Statt $Q\, x_\sigma\, \varphi$ schreibt man oft $Qx : \sigma.\, \varphi$ oder vermischt Objekt- und Metasprache und schreibt $Qx \in D_\sigma\, \varphi$, z.B. $\exists n \in \mathbb{Z}\, \varphi$ und $\forall x \in \mathbb{Z}\, \varphi$.

Für eine Interpretation $\mathcal{D}$ ist die Angabe des Bereichs D_σ der möglichen Werte von Ausdrücken der Kategorie σ eigentlich nur nötig, wenn man über diesen Bereich quantifizieren will: dann legt D_σ den Wertebereich der Belegungen $h : Var_\sigma \to D_\sigma$ fest, der für die Bedeutung der Quantoren Q_σ wichtig ist.

Wenn man über den Bereich D_σ nicht quantifizieren muß, genügt es, die Bedeutungen $c_\sigma^\mathcal{D}$ der Konstanten c_σ anzugeben, wobei natürlich c_σ ein Objekt der gewünschten Art D_σ sein sollte. Für eine Interpretation $\mathcal{D}$ der Quantorenlogik gibt man daher meist nur den Bereich an, über die die Variablen laufen, und die Bedeutungen der Konstanten, wobei die Bedeutungen „logischen Konstanten“ $\neg, \wedge, \vee, \forall, \exists$ auch noch wegegelassen werden:

$$\mathcal{D} = (U, P_1^\mathcal{D}, P_2^\mathcal{D}, \ldots, f_1^\mathcal{D}, f_2^\mathcal{D}, \ldots) \text{ mit } U = D_{Var}.$$

4.3 Natürliche Sprache mit Quantoren: QNL

Wir erweitern die Sprache NL um quantifizierte Nominalphrasen zu einer Beispielsprache QNL und geben dafür eine Interpretation an, die auf der Definiton von Quantoren aus der Quantorenlogik beruht.

In der Quantorenlogik gibt man für jede Kategorie σ, über die man quantifizieren will, eigene Quantoren $Q_\sigma : (D_\sigma \to \mathbb{B}) \to \mathbb{B}$ an. In der natürlichen Sprache quantifiziert man über viele verschiedene Bereiche, für die es keine eigene syntaktische Kategorie σ gibt, sondern die bei einer Interpretation $\mathcal{D}$ mit Hilfe von Gemeinnamen α aus dem Diskursuniversum U aller „Objekte“ durch

$$D_\alpha := \{a \in U \mid \alpha^\mathcal{D}(a) = 1\} \;\subseteq\; U,$$

also den Test auf die Eigenschaft α, ausgesondert werden.

Quantifizierte Nominalphrasen $Q\,\alpha$ wie *alle Studenten* sollten einen Quantor

$$[\![\textit{alle Studenten}]\!] : (U \to \mathbb{B}) \to \mathbb{B}$$

als Wert erhalten, der prüft, ob eine gegebene Eigenschaft $f : U \to \mathbb{B}$ auf alle Studenten zutrifft. Dazu muß die Bedeutung $Q^\mathcal{D}$ von Q aus der Eigenschaft $\alpha^\mathcal{D} : U \to \mathbb{B}$ den passenden Quantor $[\![Q\,\alpha]\!] : (U \to \mathbb{B}) \to \mathbb{B}$ berechnen, sodaß

$$Q^\mathcal{D} : (U \to \mathbb{B}) \to ((U \to \mathbb{B}) \to \mathbb{B})$$

sein muß. Statt vieler unabhängiger Quantoren $Q_\sigma^\mathcal{D}$ interpretieren wir die natürliche Sprache mit einem *„verallgemeinerten Quantor“* $Q^\mathcal{D}$, der aus jedem Gemeinnamen α einen Quantor $[\![Q\,\alpha]\!]$ berechnet, der dem speziellen Quantor $Q_\alpha^\mathcal{D}$ über den Bereich D_α entspräche, wenn α eine Kategorie wäre.

4.3.1 Die Sprache QNL mit quantifizierten Nominalphrasen

Wir erweitern die Sprache NL um zwei Kategorien, Quantoren und quantifizierte Nominalphrasen, und um zwei Satzkonstruktionen. Q steht dabei für verallgemeinerte Quantoren und QNP für Quantoren über spezielle Bereiche.

Beispiel 4.7 (vgl. Beispiel 3.4)

Syntaktische Kategorie		Metavariable und Konstruktionsmöglichkeiten	
Eigenname	PN	$e ::=$	*Anna* | *Emil* | *Maria* | ...
Gemeinname	N	$\alpha ::=$	*Frau* | *Mann* | *Student* | ...
Artikel	DET	$\delta ::=$	*die* | *der*
Verb	V	$\rho ::=$	$arbeitet^{(1)} \mid kennt^{(2)} \mid liebt^{(2)} \mid \ldots$
Konjunktion	$CONJ$	$o ::=$	$nicht^{(1)} \mid und^{(2)} \mid oder^{(2)}$
Kopula	COP	$\kappa ::=$	*ist*
Nominalausdruck	NP	$\tau ::=$	$e \mid \delta\ \alpha$
Quantor	Q	$Q ::=$	*jeder* | *jede* | *ein* | *eine*
Quant.Nomen	QNP	$\tilde{\tau} ::=$	$Q\,\alpha$
Satz	S	$\varphi ::=$	$\tau\ \kappa\ \tau \mid \tau\ \rho^{(1)} \mid \tau_1\ \rho^{(2)}\ \tau_2$
		$\mid$	$\tau_1\ \kappa\ o^{(1)}\ \tau_2 \mid \tau\ \rho^{(1)}\ o^{(1)} \mid \tau_1\ \rho^{(2)}\ \tau_2\ o^{(1)}$
		$\mid$	$\varphi_1\ o^{(2)}\ \varphi_2$
		$\mid$	$\tilde{\tau}_1\ \rho^{(2)}\ \tau_2 \mid \tilde{\tau}_1\ \rho^{(2)}\ \tau_2\ o^{(1)}$

Wir erlauben quantifizierte Nominlaphrasen *nur in der Subjektposition*, und einfachheitshalber nur in Sätzen mit transitivem Verb.

4.3.2 Die Interpretation $\mathcal{D}$ für QNL

Die Auswertung $[\![\cdot]\!]^{\mathcal{D}}$ der NL-Ausdrücke soll so angepaßt werden, daß

$$[\![(Q\,\alpha)\,\rho\,NP]\!] := Q^{\mathcal{D}}([\![\alpha]\!])(f)$$

ist für die durch $f(a) := [\![a\,\rho\,\tau]\!]$ definierte Eigenschaft $f : \texttt{person} \to \mathbb{B}$ von Personen. Bei $Q\,\alpha = $ *jeder Mann*, $\rho = $ *kennt* und $NP = $ *Maria* ist f die Eigenschaft von Personen, Maria zu kennen, und $[\![\textit{jeder Mann}]\!] := \textit{jeder}^{\mathcal{D}}([\![\textit{Mann}]\!])$ soll eine beliebige Eigenschaft f von Personen darauf testen, ob jeder Mann sie hat. Da die Nomina α Werte im Bereich $D_{PN} \to \mathbb{B}$ haben, brauchen wir

$$Q^{\mathcal{D}} : (D_{PN} \to \mathbb{B}) \to ((D_{PN} \to \mathbb{B}) \to \mathbb{B}).$$

Die Interpretation $\mathcal{D}$ aus Beispiel 3.4 erweitern wir daher wie folgt, wobei $U = D_{PN}$ das Diskursuniversum ist:

(a) Werte der verallgemeinerten Quantoren sind Funktionen, die einem Test auf Individuen (der einen Teilbereich von U festlegt), einen (über diesen Bereich laufenden) Quantor zuordnet,

$$D_Q := (U \to \mathbb{B}) \to ((U \to \mathbb{B}) \to \mathbb{B}), \qquad (21)$$

darin die wie folgt definierten Bedeutungen $\mathit{jeder}^{\mathcal{D}}$, $\mathit{ein}^{\mathcal{D}}$ der Konstanten:

$$\begin{aligned}
\mathit{jeder}^{\mathcal{D}}(f)(g) &:= \begin{cases} 1, & \text{falls für } g(a) = 1 \text{ für jedes } a \in U \text{ mit } f(a) = 1, \\ 0, & \text{sonst,} \end{cases} \\
\mathit{ein}^{\mathcal{D}}(f)(g) &:= \begin{cases} 1, & \text{falls } g(a) = 1 \text{ für mindestens ein } a \in U \text{ mit } f(a) = 1, \\ 0, & \text{sonst.} \end{cases}
\end{aligned}$$

Diese sind natürlich die Werte der Konstanten:

$$[\![\mathit{jeder}]\!] := \mathit{jeder}^{\mathcal{D}}, \qquad [\![\mathit{ein}]\!] := \mathit{ein}^{\mathcal{D}}.$$

Der unbestimmte Artikel *ein* wird hierdurch als Existenzquantor (über dem durch f ausgewählten Teilbereich von U) interpretiert. Andere Lesarten des unbestimmten Artikels, z.B. die universelle(!) Lesart in *Ein Vogel kann fliegen, ein Fisch nicht.*, werden also nicht berücksichtigt.

Die Auswertungsfunktion $[\![\cdot]\!]^{\mathcal{D}}$ muß um die quantifizierten Nominalphrasen und die Satzformen, in denen diese vorkommen dürfen, erweitert werden:

(a) Werte der quantifizierten Nominalphrasen sind Quantoren (im Sinne der Quantorenlogik):

$$[\![Q\,\alpha]\!] \quad := \quad [\![Q]\!]([\![\alpha]\!]) : (U \to \mathbb{B}) \to \mathbb{B}. \tag{22}$$

Beachte, daß der Quantor eine Eigenschaft $g : U \to \mathbb{B}$ von *beliebigen Individuen von* U darauf überprüft, ob sie für hinreichend viele $a \in U$ mit $\alpha^{\mathcal{D}}(a) = 1$ gilt.

(b) Die Auswertung von Sätzen mit quantifizierter Nominalphrase als Subjekt erfolgt durch

$$[\![\tilde{\tau}\,\rho\,\tau]\!] := [\![\tilde{\tau}]\!](a \mapsto [\![a\,\rho\,\tau]\!]), \tag{23}$$

d.h. wir extrahieren die quantifizierte Nominalpharse $\tilde{\tau}$ aus der Subjektposition des Satzes und prüfen, ob der Quantor $[\![\tilde{\tau}]\!]$ auf die Eigenschaft von Individuen zutrifft, die der Rest des Satzes als Eigenschaft eines *individuellen* Subjekts ausdrückt.

Bemerkung 4.8 Wir können diese Auswertung grob als eine *Übersetzung* $(\cdot)^{QL}$ *in die Quantorenlogik* mit speziellen Quantoren Q_α oder $Qx : \alpha$ auffassen:

$$\begin{aligned}
(Q\,\alpha\,\rho\,\tau)^{QL} &:= Qx : \alpha.\ (x\,\rho\,\tau)^{QL} \\
&:= Qx : \alpha.\ \rho^{QL}(x, \tau^{QL}).
\end{aligned}$$

Diese Übersetzung heißt in der Linguistik *quantifier-raising*. Zum Beispiel

$$(\mathit{Jeder\ Student\ kennt\ Maria})^{QL} \quad := \quad \forall x : \mathit{Student}.\ \mathit{kennt}^{QL}(x, \mathit{Maria}^{QL}).$$

Der wesentliche Punkt dabei ist, daß die Quantoren in der natürlichen Sprache in den Nominalphrasen (also quasi den Termen) stehen, in der Quantorenlogik aber Teile der Formeln, nicht der Terme sind.

4.4 Exkurs: Implementierung von QNL

Wir wollen nun diese Interpretation durch Erweitung der früheren Interpretation $\mathcal{D}$ von NL implementieren. Die Schnittstelle `INTERPRETATION` muß jetzt auch verallgemeinerte Quantoren vorsehen. Dazu dient folgene Erweiterung:

⟨*NL/qnl.sml*⟩≡

```
signature INTERPRETATION_Q =
  sig
      include INTERPRETATION

      type Quant
      val jeder : Quant
      val ein : Quant
  end
```

Die Interpretation muß den neuen Typ der verallgemeinerten Quantoren und die beiden Werte festlegen. Dazu muß der Diskursbereich $U =$ `person` *durchlaufen* werden, weshalb wir aus den Elementen des Diskursuniversums eine Liste bauen und dann eingebaute Listenoperationen von `SML` zur Definition der Werte für *jeder* und *ein* benutzen:

⟨*NL/qnl.sml*⟩+≡

```
structure Q =
  struct
    open D  (* die Interpretation von NL benutzen *)

    type Quant = (person -> bool) -> (person -> bool) -> bool

    local
        val personen = [a,b,c,d,e,f,g,h,i,j,k,l,m,n,p,q,r,s,t,u,v,w]
    in
       fun jeder F G =  List.all G (List.filter F personen)
       fun ein F G  = List.exists G (List.filter F personen)
    end
  end
```

Bemerkung 4.9 Um den Diskursbereich U zu durchlaufen, genügt seine Definition als Datentyp nicht, sondern man braucht eine *Liste* aller Objekte – das geht natürlich nur bei endlichem U. (Obwohl wir Eigenschaften absichtlich als Testfunktionen und nicht als Mengen oder Listen von Objekten implementiert haben, müssen wir hier also auf Listen zurückkommen.)

4.4.1 Grammatik und Auswertung für QNL

Es werden nur die Definitionen der Quantoren und quantifizierten Nominalphrasen sowie die beiden Satzkonstruktionen ergänzt:

⟨*NL/qnl.grm*⟩≡

```
(* Umbenennung, damit nl.grm nur erweitert werden muss: *)
structure D = Q

datatype value = object of D.PN
               | truth of bool
exception Undefined

%%
%header (functor NL_LrValsFun (structure Token : TOKEN) : NL__LRVALS)

%term
     EOF | PUNKT
   | UND | ODER | NICHT
   | DET
   | IST
   | JEDER | EIN
   | n of D.N
   | iv of D.IV
   | tv of D.TV
   | pn of D.PN

%nonterm
   start of (value * string) option
   | satz of D.S
   | np of D.NP
   | qnp of (D.PN -> D.S) -> D.S     (* nicht D.Quant *)

%start start

%eop EOF PUNKT
%noshift EOF PUNKT
%pos int

%left UND
%left ODER

%name NL_
%verbose

%%

start : satz              (SOME (truth(satz),Bool.toString satz))
     | np                 (SOME (object(np), D.toString np))
     |                    (NONE)
```

```
satz : np IST np          (D.ist(np1,np2))
     | np IST NICHT np    (D.nicht(np1 = np2))
     | np iv              (iv1(np1))
     | np iv NICHT        (D.nicht(iv1(np1)))
     | np tv np           (tv1(np1,np2))
     | np tv np NICHT     (D.nicht(tv1(np1,np2)))
     | qnp tv np          (qnp(fn x => tv(x,np)))
     | NICHT qnp tv np    (D.nicht(qnp(fn x => tv(x,np))))
     | satz UND satz      (D.und(satz1,satz2))
     | satz ODER satz     (D.oder(satz1,satz2))

np : pn                   (pn)
     | DET n              (D.der n1)

qnp: JEDER n              (D.jeder n)
     | EIN n              (D.ein n)
```

Beachte die Auswertungsregeln für die quantifizierten Nominalphrasen `qnp` und die beiden Satzformen, in denen sie vorkommen dürfen.

4.4.2 Erweiterung des Lexers

Es werden nur die Wörter *ein*, *eine* und *jeder*, *jede* ergänzt.

⟨*NL/qnl.lex*⟩≡

```
structure Tokens = Tokens
structure Interface = Interface
open Interface

type pos = Interface.pos
type svalue = Tokens.svalue
type ('a,'b) token = ('a,'b) Tokens.token
type lexresult = (svalue,pos) token

fun eof() = Tokens.EOF(!line,!line)
fun makeInt (s : string) = s

%%
%header (functor NL_LexFun(structure Tokens: NL__TOKENS
                           structure Interface: INTERFACE) : LEXER);
%full

idchars=[0-9]|[_&+/@|#*`'><];
alpha=[a-zA-ZÄÖÜäöüß];
comment=%(.*)\n;
whitespace=[\t\ ]*;
```

```
%%
<INITIAL>{whitespace} => (lex());
<INITIAL>{comment}    => (next_line(); lex());
<INITIAL>\n           => (next_line(); lex());
<INITIAL>"Der"        => (Tokens.DET(!line,!line));
<INITIAL>"Die"        => (Tokens.DET(!line,!line));
<INITIAL>"Jede"       => (Tokens.JEDER(!line,!line));
<INITIAL>"Jeder"      => (Tokens.JEDER(!line,!line));
<INITIAL>"Eine"       => (Tokens.EIN(!line,!line));
<INITIAL>"Ein"        => (Tokens.EIN(!line,!line));
"."         => (Tokens.PUNKT(!line,!line));
"und"       => (Tokens.UND(!line,!line));
"oder"      => (Tokens.ODER(!line,!line));
"nicht"     => (Tokens.NICHT(!line,!line));
"der"       => (Tokens.DET(!line,!line));
"die"       => (Tokens.DET(!line,!line));
"ist"       => (Tokens.IST(!line,!line));
"Anna"      => (Tokens.pn(D.Anna,!line,!line));
"Emil"      => (Tokens.pn(D.Emil,!line,!line));
"Maria"     => (Tokens.pn(D.Maria,!line,!line));
"Mann"      => (Tokens.n(D.Mann,!line,!line));
"Frau"      => (Tokens.n(D.Frau,!line,!line));
"Student"   => (Tokens.n(D.Student,!line,!line));
"arbeitet"  => (Tokens.iv(D.arbeitet,!line,!line));
"kennt"     => (Tokens.tv(D.kennt,!line,!line));
"liebt"     => (Tokens.tv(D.liebt,!line,!line));
"jede"      => (Tokens.JEDER(!line,!line));
"jeder"     => (Tokens.JEDER(!line,!line));
"eine"      => (Tokens.EIN(!line,!line));
"ein"       => (Tokens.EIN(!line,!line));
.           => (error ("ignoring illegal character " ^ yytext,
                       !line,!line); lex());
```

4.4.3 Erstellung des Programmpakets für QNL

Damit das Interface und der Linker vom Paket `NL` übernommen werden können, wurden im `%header` von `qnl.grm` und in `qnl.lex` die Namen der Funktoren so gelassen, wie sie für `NL` waren (dh. `NL_<...>` statt vielleicht `QNL_<...>`).

Die SML-Quelldateien `NL/qnl.sml`, `NL/qnl.grm`, `NL/qnl.lex` sowie die früher besprochenen Dateien `NL/nl.sml`, `NL/nl.interface.sml`, `NL/link.sml` und die folgende Beschreibungsdatei

⟨*NL/qnl.110.42.cm*⟩≡

```
Library
  structure Q
```

```
   structure NL
 is
   $/smlnj-lib.cm
   $/ml-yacc-lib.cm
   $/basis.cm

   nl.sml        (* Interpretationen D und Q *)
   qnl.sml
   nl.interface.sml (* fuer Lexer und Linker *)
   nl.link.sml  (* an der LMU: nl.link.110.0.3.sml *)
   qnl.lex      (* fuer Tokenizer und Parser *)
   qnl.grm
```

sind im Archiv `NL.tgz` verpackt. Damit kann man `QNL` wie folgt installieren:

(a) Man packt das Archiv mit `tar -xvfz NL.tgz` aus, geht in das entstehende Verzeichnis `NL` und startet dort SML/NJ Version 110.42 (oder eine spätere Version) mit `/usr/bin/sml`.[20]

(b) In SML ruft man den Compilation Manager mit der Beschreibungsdatei auf:

```
- CM.make "qnl.110.42.cm";
```

Das sollte zu einer Reaktion wie der folgenden führen:

```
...
[compiling (qnl.cm):nl.link.sml]
[New bindings added.]
val it = true : bool
```

Anwendung

Dann kann man den Parser und Auswerter starten (mit `NL` statt `QNL`, da wir die `%header`-Namen wie beim vorigen Beispiel benutzt haben):

```
- NL.parse();
ein Mann kennt Maria.
Value is: false
nicht jeder Student liebt Anna.
Value is: true
```

Quantifizierte Nominalphrasen in Objektstellung führen zu Syntaxfehlern, da sie in der Grammatik nicht vorgesehen sind.

[20] An der LMU-Informatik ruft man `sml-110.0.42` auf, vgl. Bemerkung 3.12.

Kurzdokumentation

Das Verzeichnis `NL` enthält folgende `README`-Datei, in der die Benutzung nochmal beschrieben ist:

⟨*NL/README*⟩≡

```
Inhalt: Beispielprogramm zur Auswertung natürlichsprachlicher Ausdrücke.
Autor:  Hans Leiß, CIS, Universität München, 9.6.2004

Systemanforderungen:

Es muß SML/NJ in einer Version > 110.42 installiert sein, ein-
schließlich der Programme mllex und mlyacc, die in der Distribution
von SML/NJ enthalten sind. (Anpassung an Version 110.0.3 siehe unten.)

Dokumentation zu SML/NJ, ML-Lex, ML-Yacc:

Online-Handbücher zu SML/NJ sowie ML-Lex und ML-Yacc findet man auf der
www-Seite der LMU/IFI/Rechnerbetriebsgruppe unter: Programmiersprachen/Sonstige.

Dateien:

Die Beispielsprache QNL mit Syntax- und Auswertungsregeln ist in qnl.grm
(im Stil von ML-Yacc) und der Tokenizer in nl.lex (im Stil von ML-Lex)
definiert.
Mit den Hilfsdateien nl.interface.sml, nl.link.sml und einer der Be-
schreibungsdateien nl.cm oder qnl.cm erzeugt der 'Compilation Manager'
CM von SML/NJ daraus einen Parser mit semantischer Auswertung.

Benutzung:

Ins Verzeichnis NL gehen, dort SML/NJ aufrufen, und darin mit dem
'Compilation Manager' CM von SML/NJ aus den .cm-Dateien das Programm
übersetzen:

Beispielsitzung:

> cd NL
> /usr/bin/sml
Standard ML of New Jersey v110.45 [FLINT v1.5], February 13, 2004
- CM.make "qnl.110.42.cm";
[autoloading]
[library $smlnj/cm/cm.cm is stable]
[library $smlnj/internal/cm-sig-lib.cm is stable]
[library $/pgraph.cm is stable]
[library $smlnj/internal/srcpath-lib.cm is stable]
```

```
[autoloading done]
[scanning qnl.cm]
[library $/ml-yacc-lib.cm is stable]
[parsing (qnl.cm):qnl.sml]
[/usr/local/smlnj/bin/ml-lex qnl.lex]

Number of states = 86
Number of distinct rows = 60
Approx. memory size of trans. table = 15360 bytes
[parsing (qnl.cm):qnl.lex.sml]
[/usr/local/smlnj/bin/ml-yacc qnl.grm]
[parsing (qnl.cm):qnl.grm.sig]
[parsing (qnl.cm):qnl.grm.sml]
[loading (qnl.cm):nl.sml]
[compiling (qnl.cm):qnl.sml]
[code: 1976, env: 2705 bytes]
[loading (qnl.cm):nl.interface.sml]
[compiling (qnl.cm):qnl.grm.sig]
[code: 56, env: 765 bytes]
[compiling (qnl.cm):qnl.grm.sml]
[code: 28014, data: 1514, env: 1547 bytes]
[compiling (qnl.cm):qnl.lex.sml]
[code: 35477, data: 17589, env: 933 bytes]
[compiling (qnl.cm):nl.link.sml]
[code: 7245, data: 173, env: 4704 bytes]
[New bindings added.]
val it = true : bool

- NL.parse();
ein Student kennt Anna.
Value is: true
der Student ist Emil.
Value is: false
Anna.
Value is: a
die Frau.
Value is: a

^D  (Abbrechen)

Nachdem die Programme einmal übersetzt sind, werden sie nur noch geladen:

- CM.make "qnl.cm";
[scanning qnl.cm]
[library $/ml-yacc-lib.cm is stable]
[loading (qnl.cm):nl.sml]
```

```
[loading (qnl.cm):qnl.sml]
[loading (qnl.cm):nl.interface.sml]
[loading (qnl.cm):qnl.grm.sig]
[loading (qnl.cm):qnl.grm.sml]
[loading (qnl.cm):qnl.lex.sml]
[loading (qnl.cm):nl.link.sml]
[New bindings added.]
val it = true : bool

- NL.parse();
jeder Mann liebt Anna.
Value is: false

^D

Nachtrag für Rechner an der LMU:
-------------------------------
Auf den Rechnern der LMU/Informatik ist unter /usr/bin/sml die
letzte 'Release'-Version, SML/NJ version 110.0.3, installiert.

Die Version 110.42 ist auch installiert, aber unter

     /usr/bin/sml-110.0.42

Sie verwendet anscheinend Bibliotheksdatein der Version 110.0.3,
weshalb man in qnl.110.42.cm die Zeile nl.link.sml durch
nl.link.110.0.3.sml ersetzen muss.

Die Funktion 'reader' aus nl.link.sml ist in nl.link.110.0.3.sml:

      fun reader (n:int) = TextIO.inputLine TextIO.stdIn

Vgl. auch die Merkblätter zur Veranstaltung 'Semantik' im SS 2004
am CIS der Universität München.                   H.Leiß, 11.6.04
```

4.5 Exkurs: Plural und mehrstellige Quantoren: PQNL

Wir wollen nun die bisherige Interpretation um die Quantoren *alle* und *einige* erweitern und zeigen, daß deren Bedeutung in stärkerem Maße vom Kontext abhängt als die der entsprechenden Quantoren *jeder* und *ein* im Singular.[21] Dazu werden wir Nomen im Plural und zweistellige Nomen einführen und außerdem Reflexiv- und Reziprokpronomen betrachten.

Zuerst wollen wir aber die Interpretation besser strukturieren. Wir haben bisher die Konjunktionen (in `nl.sml`) und die Quantoren (in `qnl.sml`) in derselben Struktur `D` bzw. `Q` definiert, in der auch die Bedeutungen der Eigennamen, Nomina und Verben erklärt war.

Da aber die Bedeutung der Konjunktionen und Quantoren normalerweise bei allen Interpretationen gleich ist, definieren wir sie besser unabhängig von der Bedeutungen der Nomina, Verben und Eigennamen in eigenen Programmmodulen, zwei SML-Strukturen `Boole` und `Quantor`. Die Bedeutungen der Eigennamen, Nomen und Verben werden dann getrennt in einer Struktur `Personen` angegeben.

Anschließend wird aus diesen drei Unterstrukturen eine Gesamtstruktur zusammengesetzt, die die Interpretation der gesamten Beispielsprache festlegt.

4.5.1 Junktoren und verallgemeinerte Quantoren

Die Bedeutung der Junktoren halten wir in einer Struktur `Bool` fest, wobei wir uns das eingebaute `bool` von SML nehmen (mit den Werten `true:bool`, `false:bool`) und um unsere eigenen Junktoren erweitern:

⟨*QNL/boole.sml*⟩≡

```
structure Boole =
  struct
    datatype bool = datatype bool (* das bool von SML, mit Werten true,false *)

    fun nicht(x) = not x
    fun und(x,y) = x andalso y
    fun oder(x,y) = x orelse y
  end
```

Die Bedeutung der Quantoren muß von dem Diskursuniversum abhängen, über das man quanitifizieren will. Es wird in der jeweiligen Interpretation angegeben durch einen (i.a. unendlichen) Typ `obj` und eine (endliche, angeordnete) Liste `objects` der Objekte daraus, über die die Quantoren laufen sollen:

⟨*QNL/quantoren.sml*⟩≡

```
signature UNIVERSUM =
  sig
```

[21] Wir hatten allerdings ignoriert, daß es auch vom Kontext abhängt, ob man *ein* überhaupt als Quantor oder nur als Artikel interpretiert.

```
    type obj
    val objects : obj list
  end
```

Die Bedeutung der Quantoren wird in einer (von einem Universum `U:UNIVERSUM`) *abhängigen Struktur* definiert, was man in SML einen Funktor nennt:

⟨*QNL/quantoren.sml*⟩+≡

```
functor Quantoren (U : UNIVERSUM) =
  struct
    type obj = U.obj
    val objects = U.objects

    exception Undefined

    (* einstellige verallgemeinerte Quantoren *)
    type Quant = (obj -> bool) -> (obj -> bool) -> bool

    fun thefirst(A) = case (List.find A objects)
                       of SOME x => x
                        | NONE => raise Undefined

    fun forall A G = List.all G (List.filter A objects)
    fun exists A G = List.exists G (List.filter A objects)

    fun most A G =
        let
            val As =  List.filter A objects
            val GAs = List.filter G As
            val nGAs = List.filter (not o G) As
        in
            (List.length GAs) > (List.length nGAs)
        end

    (* zweistellige verallgemeinerte Quantoren *)
    type QQuant = (obj -> bool)*(obj -> bool) -> (obj -> bool) -> bool

    fun more (A,B) G =
        let
            val As = List.filter A objects
            val Bs = List.filter B objects
            val GAs = List.filter G As
            val GBs = List.filter G Bs
        in
            List.length GAs > List.length GBs
        end
```

```
    (* verallgemeinerte Quantoren fuer Paare von Objekten *)

    val object_pairs =
        List.foldr (op @) []
                   (List.map (fn x => (List.map (fn y => (x,y)) objects)) objects)

    fun forall_2_2 A G = List.all G (List.filter A object_pairs)
    fun exists_2_2 A G = List.exists G (List.filter A object_pairs)
    fun forall_1_2 A G = forall A (fn x => forall A (fn y => G(x,y)))  (* ggf.: x=y or
    fun exists_1_2 A G = exists A (fn x => exists A (fn y => G(x,y)))  (* ggf.: x=/=y a
  end
```

In SML kann man mit der Deklaration

```
- structure Q = Quantoren(U);
```

eine neue Struktur `Q` bilden, die verallgemeinerte Quantoren enthält, die sich auf das Universum `U` beziehen: z.B. wird dann `(most A G)` durch „die meisten Objekte *aus U* mit der Eigenschaft `A` erfüllen `G`“ interpretiert.

4.5.2 Das Diskursuniversum der Personen

Das Diskursuniversum und die Bedeutungen der Individuennamen, Nomen und Verben wird nun in einer Struktur `Personen` festgelegt. Verallgemeinerte Quantoren, die sich auf das Universum der Personen beziehen, werden mit Hilfe des oben definierten Funktors `Quantoren` interpretiert.

Er enthält neben den einstelligen verallgemeinerten Quantoren, mit denen wir u.a. *jeder* und *ein* interpretieren, auch zwei verallgemeinerte Quantoren für Paare von Objekten, und zwar All- und Existenzquantoren, $\forall A\, G$ und $\exists A\, G$, die eine zweistellige Eigenschaft G an den Paaren einer Relation A oder der aus einer Menge A gebildeten Relation $A \times A$ prüfen. Mit ihnen werden *alle*, *einige*, *je_zwei* und *ein_Paar* interpretiert.

⟨*QNL/personen.sml*⟩≡

```
structure Personen =
  struct
    datatype person = a | b | c | d | e | f | g | h | i | j | k
                    | l | m | n | p | q | r | s | t | u | v | w

    structure Q =
      Quantoren(struct type obj = person
                       val objects = [a,b,c,d,e,f,g,h,i,j,k,l,m,n,p,q,r,s,t]
                end)

    (* Werte der verallgemeinerten Quantoren ueber Personen *)

    val ein   = Q.exists
```

```
val jeder = Q.forall
val der   = Q.thefirst

val alle    = Q.forall_2_2
val einige  = Q.exists_2_2
val je_zwei = Q.forall_1_2
val ein_paar = Q.exists_1_2

(* Werte der Eigennamen und der ein- und zweistelligen Nomen *)

val Anna = a
val Dieter = d
val Emil = e
val Maria = m

fun Frau(a | b | c | g | h | i | m | n | s | t | u) = true
  | Frau(_) = false

fun Mann(d | e | f | j | k | l | p | q | r | v | w) = true
  | Mann(_) = false

fun Student(u | v | w) = false
  | Student(_) = true

fun Bruder((d,e) | (e,d)) = true
  | Bruder(_,_) = false

fun Nachbar((a,d) | (a,e) | (b,a) | (b,c) | (d,e)) = true
  | Nachbar(_,_) = false

(* Werte der ein- und zweistelligen Verben *)

fun arbeitet(a | b | c | d | e) = true
  | arbeitet(_) = false

fun kennt((a,e) | (a,m) | (e,a) | (m,e)) = true
  | kennt(x,y) = if x=y then true else false

fun liebt((e,m) | (a,e) | (d,e) | (e,d)) = true
  | liebt(x,y) = false

fun ist(x,y) = if x=y then true else false

val toString = (fn a => "a" | b => "b" | c => "c"
                 | d => "d" | e => "e" | m => "m"
                   | _ => "some person")
```

```
end
```

4.5.3 Interpretation von PQNL

Die Interpretation definieren wir nun in einer Struktur `D` mit zwei Unterstrukturen `B` und `P` und den Wertebereichen der syntaktischen Kategorien.

Die Unterstruktur `B` enthält den Bereich der Wahrheitswerte mit den Bedeutungen der Junktoren. Die Unterstruktur `P` enthält den Bereich der Personen mit den Bedeutungen der Eigennamen, Nomen, Verben und den über die Personen laufenden Quantoren. Die Werte der Konstanten werden in den Auswertungsregeln den beiden Unterstrukturen entnommen.

⟨*QNL/pqnl.sml*⟩≡

```
structure D =
  struct
     structure B = Boole
     structure P = Personen

     type person = Personen.person

     (* Wertebereiche der syntaktischen Kategorien: *)

     type PN = person
     type NP = person
     type S = bool
     type N = person -> bool
     type N2 = person * person -> bool
     type IV = person -> bool
     type TV = person * person -> bool
  end
```

Die Kategorie der Kopula ist weggelassen, da wir die Kopula im Plural anders interpretieren als im Singular.

4.5.4 Grammatik und Auswertung für PQNL

Die Grammatik soll nun um Ausdrücke erweitert werden, bei denen es auf den Unterschied zwischen Singular und Plural ankommt. Es geht uns hier nicht um die Kongruenz zwischen Nomen und Verb, sondern darum, wie der Numerus des Subjekts und die Stelligkeit des Verbs zusammenspielen und wie die Bedeutung der Quantoren im Plural von der Stelligkeit des Nomens, durch den der Quantor beschränkt wird, beeinflußt wird.

Wir wollen Aussagen wie *Einige Studenten lieben einander*, *Jeder Student kennt sich*, *Alle Studenten kennen einander*, *Einige Nachbarn sind Brüder*, *Alle*

Menschen sind Brüder interpretieren. Hier hängt einerseits die Bedeutung der Quantoren von der Stelligkeit des folgenden Nomens ab, und andererseits werden wesentliche Eigenschaften des Verbs durch die Verwendung der Pronomen verändert.

Mit dem Reflexivpronomen werden aus transitiven Verben „syntaktische" intransitive Verben gebildet, die wir von den lexikalischen intransitiven Verben `ivl` unterscheiden. Mit dem Reziprokpronomen weren aus transitiven Verben symmetrische transitive Verben, für die eine Kategorie `stv` eingeführt wird.

⟨*QNL/pqnl.grm*⟩≡
```
structure B = D.B
structure P = D.P

datatype value = object of D.PN | truth of bool
exception Undefined

%%
%header (functor NL_LrValsFun (structure Token : TOKEN) : NL__LRVALS)

%term
     EOF | PUNKT
   | UND | ODER | NICHT
   | DET
   | IST | SIND
   | pn of D.PN
   | n of D.N
   | n2 of D.N2
   | ivl of D.IV  (* lexikalisch intransitive Verben *)
   | tv of D.TV
   | JEDER | EIN | ALLE | EINIGE
   | refl
   | rezi

%nonterm
   start of (value * string) option
   | satz of D.S
   | np of D.NP
   | qnp of D.N -> bool
   | qnp2 of D.N2 -> bool
   | iv of D.IV
   | stv of D.TV

%start start

%eop EOF PUNKT
```

```
%noshift EOF PUNKT
%pos int

%left UND
%left ODER

%name NL_
%verbose

%%

start : satz              (SOME (truth(satz), Bool.toString satz))
     | np                 (SOME (object(np), P.toString np))
     |                    (NONE)

satz : np IST np          (P.ist(np1,np2))
     | np IST NICHT np    (B.nicht(P.ist(np1,np2)))
     | np iv              (iv(np))
     | qnp iv             (qnp(fn x => iv(x)))
     | np iv NICHT        (B.nicht(iv(np)))
     | np tv np           (tv(np1,np2))
     | np tv np NICHT     (B.nicht(tv(np1,np2)))
     | qnp tv np          (qnp(fn x => tv(x,np)))
     | NICHT qnp tv np    (B.nicht(qnp(fn x => tv(x,np))))
     | satz UND satz      (B.und(satz1,satz2))
     | satz ODER satz     (B.oder(satz1,satz2))
     | qnp2 stv           (qnp2 stv)
     | qnp2 SIND n2       (qnp2 n2)

np : pn                   (pn)
     | DET n              (P.der n)

qnp: JEDER n              (P.jeder n)
     | EIN n              (P.ein n)

qnp2 : ALLE n2            (P.alle n2)
     | EINIGE n2          (P.einige n2)
     | ALLE n             (P.je_zwei n)
     | EINIGE n           (P.ein_paar n)

iv : ivl                  (ivl)
     | tv refl            (fn x => tv(x,x))

stv : tv rezi             (fn (x,y) => B.und(tv(x,y),tv(y,x)))
     | SIND n             (fn (x,y) => B.und(n(x),n(y)))
```

Die Quantoren `ALLE` und `EINIGE` werden nicht durch den Lexer interpretiert, sondern *kontextabhängig* in der Grammatik: je nachdem, ob das folgende Nomen ein- oder zweistellig ist, wird eine andere Bedeutung gewählt.

Aufgabe 4.1 Wir haben die Kopula im Singular, `IST`, durch die Gleichheit von Individuen interpretiert, und im Plural, `SIND`, durch die Teilmengenrelation. In Sätzen wie *Jede Frau ist ein Student* und *die Schuldigen sind Dieter und Maria* braucht man aber gerade die umgekehrte Interpretation, sodaß man auch die Kopula kontextabhängig interpretieren muß.

Man führe eine Regel ein, nach der die Kopula `IST` in Sätzen wie *Jede Frau ist ein Student* durch die Elementbeziehung (und *ein* nicht als Existenzquantor!) interpretiert wird.

Die angegebenen Auswertungsregeln stimmen sicher nicht in allen Einzelheiten. Eine korrekte Darstellung muß sehr viele unterschiedliche Kombinationen von Singular und Plural, Verwendung und Weglassung des Artikels, Lesarten des unbestimmten Artikels als Quantor oder Artikel, Reflexiv- und Reziprokpronomen, einstelligen und zweistelligen Nomen und Verben, berücksichtigen. Es ist keineswegs klar, daß man diese einzelnen syntaktischen Elemente bei der Auswertung unabhängig von einander behandeln kann (oder sollte).

Mit so einfachen Kategorien wie NP und VP ist es jedenfalls nicht getan. Es ist an den bisherigen Beispielen aber auch nicht abzusehen, ob man bei einer feineren Aufgliederung der syntaktischen Kategorien durch Merkmale eine rekursive Definition der Auswertung angeben kann, die nur die Kategorien der direkten Teilausdrücke verwendet. Vielleicht ist es zweckmäßiger, mit der Auswertungsfunktion „tiefer“ in die Struktur der Ausdrücke „hineinzuschauen“, anstatt alle relevanten Informationen über die Teilausdrücke in Merkmalen ihrer Kategorien zu kodieren; z.B. läßt sich Information über die *Beziehung* zwischen Teilausdrücken nur künstlich auf die einzelnen Teilausdrücke verteilen.

Es ist a priori ja nicht einmal klar, daß sich syntaktische Gruppierungen von Wörtern, die sich z.B. durch Kriterien der gemeinsamen Verschiebbarkeit im Satz ergeben, überhaupt eine sinnvolle Bedeutungseinheit bilden.

4.5.5 Lexer für PQNL

Für jedes erlaubte Wort sehen wir ein Token vor, das i.a. den Wert des Worts bei der Interpretation `D` enthält. (Natürlich müssen Lexer und Parser dieselbe Struktur `D` als Interpretation verwenden; andernfalls meldet SML Typfehler.)

Ob der Lexer ein Wort direkt an den Parser weitergeben oder ob er es auswerten und den Wert weitergeben sollte, hängt davon ab, ob das Wort mehrdeutig ist und man erst am syntaktischen Kontext die passende Bedeutung erkennen kann oder ob es eine eindeutige, kontextunabhängige Bedeutung hat.

$\langle$*QNL/pqnl.lex*$\rangle \equiv$

```
structure P = D.P

structure Tokens = Tokens
structure Interface = Interface

open Interface

type pos = Interface.pos
type svalue = Tokens.svalue
type ('a,'b) token = ('a,'b) Tokens.token
type lexresult = (svalue,pos) token

fun eof() = Tokens.EOF(!line,!line)
fun makeInt (s : string) = s

%%
%header (functor NL_LexFun(structure Tokens: NL__TOKENS
                           structure Interface: INTERFACE) : LEXER);
%full

idchars=[0-9]|[_&+/@|#*`'><];
alpha=[a-zA-ZÄÖÜäöüß];
comment=%(.*)\n;
whitespace=[\t\ ]*;
%%
<INITIAL>{whitespace} => (lex());
<INITIAL>"Der"    => (Tokens.DET(!line,!line));
<INITIAL>"Die"    => (Tokens.DET(!line,!line));
<INITIAL>"Jede"   => (Tokens.JEDER(!line,!line));
<INITIAL>"Jeder"  => (Tokens.JEDER(!line,!line));
{comment}  => (next_line(); lex());
\n         => (next_line(); lex());
"."        => (Tokens.PUNKT(!line,!line));
"und"      => (Tokens.UND(!line,!line));
"oder"     => (Tokens.ODER(!line,!line));
"nicht"    => (Tokens.NICHT(!line,!line));
"der"      => (Tokens.DET(!line,!line));
"die"      => (Tokens.DET(!line,!line));
"ist"      => (Tokens.IST(!line,!line));
"Anna"     => (Tokens.pn(P.Anna,!line,!line));
"Dieter"   => (Tokens.pn(P.Dieter,!line,!line));
"Emil"     => (Tokens.pn(P.Emil,!line,!line));
"Maria"    => (Tokens.pn(P.Maria,!line,!line));
"Mann"     => (Tokens.n(P.Mann,!line,!line));
"Frau"     => (Tokens.n(P.Frau,!line,!line));
"Student"  => (Tokens.n(P.Student,!line,!line));
```

```
"arbeitet" => (Tokens.ivl(P.arbeitet,!line,!line));
"kennt"    => (Tokens.tv(P.kennt,!line,!line));
"liebt"    => (Tokens.tv(P.liebt,!line,!line));
"jede"     => (Tokens.JEDER(!line,!line));
"jeder"    => (Tokens.JEDER(!line,!line));
"eine"     => (Tokens.EIN(!line,!line));
"ein"      => (Tokens.EIN(!line,!line));
"alle"     => (Tokens.ALLE(!line,!line));
"einige"   => (Tokens.EINIGE(!line,!line));
"sich"     => (Tokens.refl(!line,!line));
"einander" => (Tokens.rezi(!line,!line));
"Nachbarn" => (Tokens.n2(P.Nachbar,!line,!line));
"Brüder"   => (Tokens.n2(P.Bruder,!line,!line));
"Männer"   => (Tokens.n(P.Mann,!line,!line));
"Frauen"   => (Tokens.n(P.Frau,!line,!line));
"Studenten"  => (Tokens.n(P.Student,!line,!line));
"kennen"   => (Tokens.tv(P.kennt,!line,!line));
"lieben"   => (Tokens.tv(P.liebt,!line,!line));
"sind"     => (Tokens.SIND(!line,!line));
.          => (error ("ignoring illegal character " ^ yytext,
                      !line,!line); lex());
```

4.5.6 Erstellung des Programmpakets für PQNL

Zur Installation packt man das Archiv `QNL.tgz` mit `tar -xvfz QNL.tgz` aus, geht in das entstandene Verzeichnis `QNL` und startet dort `/usr/bin/sml`[22]. In SML ruft man dann auf:

```
- CM.make "pqnl.110.42.cm";
```

Die Beschreibungsdatei zum Erstellen des Pakets ist:

⟨*QNL/pqnl.110.42.cm*⟩≡

```
Library
  structure D
  structure NL
is
  $/smlnj-lib.cm
  $/ml-yacc-lib.cm
  $/basis.cm

  boole.sml
  quantoren.sml
  personen.sml
  pqnl.sml            (* Interpretationen D *)
```

[22]bzw. `/usr/bin/sml-110.0.42` an der LMU, vgl. Bemerkung 3.12

```
nl.interface.sml (* fuer Lexer und Linker *)
nl.link.sml   (* an der LMU: nl.link.110.0.3.sml *)
pqnl.lex      (* fuer Tokenizer und Parser *)
pqnl.grm
```

4.5.7 Beispiele

Folgende Beispiele zeigen die kontextabhängige Bedeutung der Quantoren im Plural und die Bedeutung der Kopula und des Reflexiv- und Reziprokpronomens.

(a) Quantifizierung mit einstelligem Nomen (im Singular) und zweistelligen Verb, das durch ein Reflexivpronomen einstellig wird:

⟨*Beispiel*⟩≡

```
- NL.parse();
ein Student kennt sich.
Value is: true
jeder Student kennt sich.
Value is: true
```

(b) Quantifizierung mit einstelligem Nomen (im Plural) und transitivem Verb, das durch ein Reziprokpronomen symmetrisch wird:

⟨*Beispiel*⟩+≡

```
einige Studenten kennen einander.
Value is: true
alle Studenten kennen einander.
Value is: false
```

(c) Quantifizierung mit zweistelligem Nomen und transitivem Verb, das durch ein Reziprokpronomen symmetrisch wird:

⟨*Beispiel*⟩+≡

```
alle Nachbarn kennen einander.
Value is: false
alle Brüder lieben einander.
Value is: true
```

(d) Quantifizierung mit einstelligem Nomen (im Plural) und einstelligem Prädikatsnomen: *Einige Studenten sind Frauen*

⟨*Beispiel*⟩+≡

```
alle Frauen sind Studenten.
Value is: true
alle Studenten sind Frauen.
Value is: false
```

(e) Quantifizierung mit zweistelligem Nomen (im Plural) und zweistelligem (symmetrischem) Prädikatsnomen:

$\langle$*Beispiel*$\rangle{+}{\equiv}$
```
einige Nachbarn sind Brüder.
Value is: true
alle Brüder sind Nachbarn.
Value is: false
```

(f) Quantifizierung mit einstelligem Nomen (im Plural) und zweistelligem (symmetrischem) Prädikatsnomen:

$\langle$*Beispiel*$\rangle{+}{\equiv}$
```
einige Studenten sind Brüder.
Value is: true
alle Studenten sind Nachbarn.
Value is: false
```

Aufgabe 4.2 Man erweitere die Grammatik so, daß quantifizierte Nominalphrasen auch in der Objektposition erlaubt sind. Sind sowohl die Subjekt- als auch die Objektnominalphrase quantifiziert, so lege man durch die Auswertungsregel fest, daß der Quantor der Subjektnominalphrase den weiteren Wirkungsbereich hat. Z.B. soll *Jeder Mann liebt eine Frau* also nicht bedeuten: *Es gibt eine Frau, die jeder Mann liebt.*

Erlaubt man Subjekt- oder Objektnominalphrasen, die aus einem zweistelligen Nomen aufgebaut sind, nicht nur bei symmetrischen transitiven Verben, so muß man sich bei der Auswertung für eine der Lesarten entscheiden: soll z.B. *Emil kennt alle Nachbarn* bedeuten, daß er alle Personen kennt, die *seine* Nachbarn sind, oder daß er alle Personen kennt, die Nachbarn von einander sind?

Bemerkung 4.10 Unsere Behandlung der Quantoren setzt eine Interpretation $\mathcal{D}$ mit eindeutigem (endlichem) Diskursuniversum D voraus, aus dem bei der Auswertung $(Q\,x \in N\,\varphi)^{\mathcal{D}} = Q^{\mathcal{D}}(\{a \in N^{\mathcal{D}} \mid \mathcal{D} \models \varphi(a)\})$ die Teilbereiche $N^{\mathcal{D}} \subseteq D$ ausgesondert werden. Daher konnten wir die Definition der $Q^{\mathcal{D}}$ in der (Unter-)Struktur der Personen festlegen und die Auswertungsregel so formulieren:

$\langle$*Auswertung von Quantoren*$\rangle{\equiv}$
```
qnp : JEDER n                  (P.jeder n)
```

Man braucht aber auch *mehrsortige* Interpretationen $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \ldots)$ mit verschiedenen gleichrangigen Diskursuniversen $D_1, D_2, \ldots$, z.B. körperliche Gegenstände und Abstrakta. Leider sieht man am syntaktischen Ausdruck $Qx \in N\,\varphi$ nicht, aus welchem der Bereiche $N^{\mathcal{D}}$ ausgewählt werden muß. Man würde gerne die Auswertung vom Typ des Nomens $N : D_i \to \mathbb{B}$ abhängig machen, etwa

$\langle$*Typabhängige Auswertung von Quantoren*$\rangle{\equiv}$
```
qnp : JEDER (n:P -> bool)      (jeder P.Q.objects n)
```

Dann könnte man ein einziges $Q^{\mathcal{D}}$ definieren und nicht verschiedene $Q^{\mathcal{D}_i}$. (Würde man stattdessen zu jedem Nomen $N : D_i \to \mathbb{B}$ eine Liste seiner Elemente angeben, so würde bei vielen $N \subseteq N'$ die Struktur $\mathcal{D}_i$ erheblich vergrößert.)

Für mehrsortige Interpretationen muß die Syntaxanalyse zumindest die Kategorie der Nomina aufspalten und jeder Unterkategorie ein Universum D_i zuordnen, sodaß man in der Auswertungsregel passende Quantoren $Q^{\mathcal{D}_i}$ angeben kann.

5 Situationsabhängige Bedeutung

Die in der Mathematik verwendeten Sprachen der Prädikaten- und Quantorenlogik genügen, um zeit- und ortsunabhängige Sachverhalte auszudrücken. In den Fällen, wo man über zeit- oder räumliche Verhältnisse sprechen muß, „vergegenständlicht" man Zeit und Raum in ideale Zeit- und Raum*punkte* und die Abhängigkeit der Verhältnisse zu *Funktionen* von Zeit- und Raumpunkten. Dann kann man über diese Zeit- und Raumpunkte sowie die Funktionen wie über statische Objekte mathematische, zeit- und ortsunabhängige Aussagen machen.

Wenn man etwa in der Physik die Temperatur eines Körpers und seine Bewegung im Raum beschreiben will, stellt man die Zeit als $\mathcal{T} = \mathbb{R}$ und den Raum als $\mathcal{R} = \mathbb{R}^3$ dar und Ort und Temperatur des Körpers als Funktionen $O : \mathcal{T} \to \mathcal{R}$ und $T : \mathcal{T} \to \mathit{Temp}$ dar. Der Körper selbst, d.h. alles, was man von ihm wissen muß, wird durch das Paar (O, T) der Wertverläufe seines Orts und seiner Temperatur dargestellt. Man betrachten sozusagen „von außerhalb" die gesamte Veränderung von Temperatur und Ort des Körpers „im Laufe der Zeit".

Die natürlichen Sprachen werden üblicherweise ganz anders verwendet: man befindet sich in einer gewissen Situation und macht Aussagen über die Verhältnisse in dieser Situation – es ist meistens unmöglich, über die sich mit der Situation ändernden Verhältnisse viel zu wissen. Daher spielen situationsunabhängige Aussagen über die Änderung der Verhältnisse „im Laufe der Situationswechsel" eine geringere Rolle; viel wichtiger sind situationsabhängige Aussagen „über" die Situation, in der man sich gerade befindet, oder eine bestimmte andere Situation.[23]

Unter den „Objekten" des Diskursuniversums gibt es sehr verschiedene Arten, über die auch in verschiedener Weise gesprochen wird. Darunter sind

(a) körperliche Gegenstände, die einen Raum einnehmen, eine Form haben, aus einem Material bestehen, viele zeitunabhängige (inhärente) und einige zeitabhängige Eigenschaften haben, und als zählbare Einheiten (Individuen) vorkommen,

(b) Lebewesen und andere körperliche Gegenstände, die zusätzlich zeitabhängige „innere Zustände" haben, z.B. müde sind oder defekt,

(c) Materialien (Arten von Materie), die Aggregatzustände haben können und in meßbaren Mengen, aber nicht in zählbaren Einheiten vorkommen, z.B. Gesteine, Flüssigkeiten, Gase,

(d) Zeit und Raum, über die ebenfalls in meßbaren Mengen, nicht in zählbaren Einheiten, und indirekt mit Hilfe von Ereignissen oder körperlichen Gegenständen und Orten gesprochen wird,

[23] Es ist aber problematisch, sich die Situationen als atomare Objekte (wie Zeitpunkte in der Physik) vorzustellen, da z.B. eine Situation eine Teilsituation einer anderen sein kann.

(e) „äußere" Zustände, d.h. zeitabhängige Verhältnisse der Umgebung, in der gesprochen wird, z.B. Zustände des Wetters, und zeitunabhängige Eigenschaften des Orts,

(f) Ereignisse, d.h. zeitliche Änderungen von Verhältnissen, die –vermutlich weil sie überschaubar sind– irgendwie als Einheiten verstanden werden,

(g) geistige Gegenstände wie Sachverhalte, Regelmäßigkeiten, Gedanken, die untereinander oder mit materiellen Gegenständen oder Lebewesen in bestimmten Beziehungen stehen, z.B. aus einander folgen, auf etwas zutreffen, oder von jemandem geglaubt oder vermutet werden.

Über diese Arten von „Objekten" wird mit teilweise unterschiedlichen Ausdrücken gesprochen und die Vorstellung von „Objekt" im Sinne einer zählbaren Einheit ist nicht in allen Fällen angebracht.

Eine Untersuchung der Bedeutung natürlichsprachlicher Ausdrücke sollte also nicht nur die zeitunabhängigen Beziehungen zwischen körperlichen Gegenständen erfassen, sondern auch Ausdrücke α, deren Bedeutung *sitationsabhängig* ist. Sie erhalten einen Wert $[\![\alpha]\!]_s^{\mathcal{D},h}$ *relativ zu einer Situation s*. Was eine Situation s ist, lassen wir offen; es könnte z.B. einfach ein Zeitpunkt sein.

Um eine Auswertung $[\![\cdot]\!]_s^{\mathcal{D}}$ zu definieren, die gegebene Situationen s berücksichtigt, müssen im Allgemeinen auch die Bedeutungen der Konstanten c relativiert werden, d.h. statt $c^{\mathcal{D}}$ brauchen wir relative Varianten $c_s^{\mathcal{D}}$.

Manche Konstanten, z.B. die logischen Konstanten $\wedge, \rightarrow$ usw., sollen aber wie bisher in $\mathbb{B}$ interpretiert werden, d.h. es genügt, die bisherige Bedeutung $\rightarrow^{\mathcal{D}}:=\rightarrow^{\mathbb{B}}$ unverändert zu lassen und nur die Auswertungsfunktion zu

$$[\![\rightarrow]\!]_s^{\mathcal{D}} \quad := \quad \rightarrow^{\mathbb{B}} \tag{24}$$

anzupassen. Dasselbe kann man bei jeder Konstanten machen, deren Bedeutung bei allen betrachteten Situationen s gleich sein soll.

Oft nimmt man an, daß die Bedeutung der Namen c von Individuen des Diskursuniversums U sich nicht mit der Situation ändert, sondern höchstens die Eigenschaften und Beziehungen zwischen den Individuen. (Z.B. hat Aristoteles angenommen, daß die durch Verben ausgedrückten Eigenschaften und Beziehungen mit der Zeit variieren, die durch Nomina ausgedrückten aber nicht.) Dann sollte die die Bedeutung der Individuenkonstanten c *unabhängig* von einer Situation s definiert werden, also

$$c^{\mathcal{D}} \quad \in \quad U,$$

die der Prädikate *relativ* zu einer Situtation s definiert werden:

$$P_s^{\mathcal{D}} \quad : \quad U^n \to \mathbb{B} \qquad \text{oder} \qquad P^{\mathcal{D}} : S \to (U^n \to \mathbb{B}),$$

wenn S der festgelegte Bereich von Situationen ist.

Wir betrachten zuerst Sprechweisen über Zeit und Ort, die mit der Quantifizierung zusammenhängen, und anschließend einfache Aspekte der Sprechsituation.

Über Zeit und Raum spricht man in der natürlichen Sprache anders als in der Physik: wir unterstellen z.B. kein festes Koordinatensystem, in dem jeder Punkt durch seine Lage relativ zum Ursprung des Koordinatensystems festgelegt ist.

Stattdessen verwenden wir sprachliche Ausdrücke, die Zeit- und Ortsangaben *relativ zu Zeit und Ort der Äußerung* machen, etwa *jetzt* oder *in drei Tagen* oder *damals, als ...*, und entsprechend *hier*, *in 3 km Entfernung* oder *dort, wo ...*, und verwenden dabei unterschiedliche Einheiten, z.B. Stunden-, Tages- oder Jahreseinheiten, Meter, Zentimeter oder Schritte, sowie unterschiedliche Koordinatensysteme, z.B. bei Entfernungsangaben entlang bekannter Wege oder entlang der „Luftlinie“

Syntaktisch werden Zeit- oder Ortsangaben durch *Adverbiale* gemacht. Ihr Wert ist eine (nicht unbedingt punktförmige) Stelle in der Zeit oder dem Raum. Über diese Stellen kann man auch quantifizieren, etwa in *immer = zu jeder Zeit* oder *nirgends = an keinem Ort*. Von den Zeit- oder Ortsangaben sind die Angaben zu Abständen von Stellen in Zeit und Raum zu unterscheiden, also Angaben zur Dauer von Ereignissen oder zu Entfernungen oder Umbegungen von Orten.

5.1 Adverbiale Bestimmungen der Zeit

Im Vergleich zu den sinnlich wahrnehmbaren Gegenständen ist die Zeit weniger „direkt“ gegeben. Wir beobachten etwa folgende Darstellung der Zeit in den Ausdrucksweisen der natürlichen Sprache:

(a) Zeitpunkte und Zeiträume sind nur durch Ereignisse unterscheidbar, die „zu dieser Zeit“ stattfinden oder nicht stattfinden. Zeiträume werden mit Hilfe von Ereignissen benannt, die diesen Zeitraum „einnehmen“, etwa *während des 30-jährigen Krieges*, oder „nicht einnehmen“, etwa *nachdem Anna gegangen war*. Zeitpunkte werden als Anfangs- oder Endzeitpunkte von Ereignissen aufgefaßt, etwa *als die Uhr 12 schlug*.

(b) Zeiträume haben eine *meßbare* Ausdehnung, die *Dauer* von Ereignissen.

    Einerseits kann die Ausdehnung eines Zeitraums als ein Vielfaches der Dauer eines Standardereignisses, einer *Einheitsdauer*, angegeben werden. Als Standardereignisse werden regelmäßig auftretende natürliche Ereignisse benutzt, wie *ein Tag* als die Zeit von Sonnenaufgang bis -Untergang, oder künstlich hervorgerufene Ereignisse wie die Bewegungen eines Uhrzeigers in *eine Minute*. Als Maßeinheit der Dauer dient der Gemeinname eines Standardereignisses.

    Andererseits kann die Ausdehnung von Zeiträumen miteinander verglichen werden, ohne Bezug auf ein Standardereignis, etwa in *es dauert länger, diese Seite abzuschreiben, als sie zu kopieren.*

(c) Zeiträume sind *eindimensional*, und die Zeitpunkte in einem Zeitraum sind relativ zu einander *angeordnet*: der Anfangs(zeit)punkt eines Ergeignisses liegt *vor* seinem End(zeit)punkt.[24]

Durch *früher*, *jetzt* und *später* wird die Zeit in drei Zeiträume eingeteilt: der Zeitraum der Vergangenheit liegt (punktweise) vor dem der Zukunft, und die Gegenwart erscheint als (einziger) Zeitpunkt dazwischen.

(d) Zwischen Zeitpunkten gibt es einen *Abstand*, die Dauer von Ereignissen, die den ersten zum Anfangs- und den zweiten zum End(zeit)punkt haben.

Mit Hilfe des Zeitabstands können Zeitpunkte relativ zu einem benannten Zeitpunkt bezeichnet werden: *gestern vor vier Wochen*, *drei Tage nach Weihnachten*, *(ab jetzt) in zehn Minuten.*

(e) Die Zeit erscheint in der Sprache *diskret* dargestellt zu sein: zu jedem Zeitpunkt gibt es einen *letzten Moment* vorher und den *nächsten Augenblick* nachher. Wir benutzen $t - 1$ und $t + 1$ als Bezeichnungen für den dem Zeitpunkt t unmittelbar vorhergehenden bzw. nachfolgenden Zeitpunkt.

(f) Über Zeitpunkte und Zeiträume kann quantifiziert werden: *irgendwann*, *nie*, *immer*, *vor langer Zeit*, *in vielen Jahren*, *an manchen Tagen.* (Das *jedesmal, wenn* φ ist eher eine Quantifizierung über die logisch möglichen Situationen, in denen φ gilt, nicht über die Zeitpunkte, wo φ gilt.)

Aus diesen Gründen stellen wir die Zeit als eine lineare Anordnung von (unteilbaren) Zeitpunkten dar, die weder einen Anfang noch ein Ende hat. Dabei benutzen wir –im Unterschied zu den Physikern– als Gesamtheit der Zeitpunkte nicht den Bereich $\mathbb{R}$ der reellen Zahlen (in dem zwischen zwei verschiedenen Punkten stets ein weiterer liegt), sondern den Bereich $\mathbb{Z}$ der ganzen Zahlen (wo zwischen t und $t + 1$ nichts liegt).

Daher interpretieren wir die Zeit durch eine diskrete Anordnung

$$\mathcal{T} = (\mathbb{Z}, <, c_1, c_2, \ldots) \tag{25}$$

mit bestimmten Zeitpunkten $c_1, c_2, \ldots$, für die es eine sprachliche Benennung gibt, etwa c_1 für den Zeitpunkt der Geburt einer bedeutenden Person, der als Anfangspunkt z.B. der christlichen Zeitrechnung genommen wird.

5.1.1 Die Zeitlogik TL mit Temporaloperatoren

Bevor wir die Zeitabhängigkeit in die Bedeutung natürlichsprachlicher Ausdrücke einbauen, geben wir eine logische Sprache mit zeitabhängiger Interpretation an. Dabei wollen wir die (Junktoren, Quantoren und) Terme zeitunabhängig, aber die Formeln zeitabhängig interpretieren.

[24] Der Begriff des Ereignisses ist nicht sehr deutlich. Muß ein Ereignis erkennbare Endpunkte haben? Wenn nicht – wird dann der Unterschied zwischen Ereignis und Zustand –wo „nichts passiert“– unklar? Kann ein Ereignis Teilereignisse haben? Wann ist es eine von anderen Ereignissen trennbare Einheit?

Durch Terme sollen nur Individuen aus dem Diskurs-Universum U bezeichnet werden, aber nicht die Zeitpunkte. In den Formeln kommt der zeitliche Aspekt, der in der natürlichen Sprache durch das Tempus von Verben und durch Zeitadverbiale ausgedrückt wird, nur durch *Temporaloperatoren* zum Ausdruck.

Beispiel 5.1 Die Sprache TL der Zeitlogik erweitert die Sprache QL der Quantorenlogik um die syntaktische Kategorie der (ein- oder zweistelligen) Temporaloperatoren und weitere Möglichkeiten beim Formelaufbau:

Synt. Kategorie	Metavariable und Konstruktionsmöglichkeiten	
Temporaloperator	$O ::=$	$F^{(1)} \mid G^{(1)} \mid P^{(1)} \mid H^{(1)} \mid S^{(2)} \mid U^{(2)}$
Formel	$\varphi ::=$	$\ldots \mid O^{(1)}\varphi \mid O^{(2)}(\varphi_1, \varphi_2)$

5.1.2 Interpretation von TL

Eine Interpretation $\mathcal{D} = (\mathcal{T}, \mathcal{D}', F^{\mathcal{D}}, \ldots, U^{\mathcal{D}})$ für TL besteht aus

(a) der Zeitstruktur $\mathcal{T} = (\mathbb{Z}, <^{\mathbb{Z}}, +^{\mathbb{Z}}, \ldots)$

(b) eine Interpretation $\mathcal{D}' = (U, \rho^{\mathcal{D}'}, \ldots, f^{\mathcal{D}'}, \ldots)$ für QL, außer daß die n-stelligen Prädikate $\rho^{(n)}$ durch *zeitabhängige* Tests auf Individuen

$$\chi : \mathbb{Z} \to (U^n \to \mathbb{B}),$$

interpretiert werden, die jedem Zeitpunkt $t \in \mathbb{Z}$ einen Test $\chi(t) : U^n \to \mathbb{B}$ zuordnen,

(c) einer Bedeutung der Temporaloperatoren, die *Test* $f, g : \mathbb{Z} \to \mathbb{B}$ *auf Zeitpunkten wieder Tests auf Zeitpunkten* zuordnen:

(a) $F\varphi$ soll die Bedeutung von φ *wird einmal der Fall sein* erhalten:

$$F^{\mathcal{D}}(f)(t) := \begin{cases} 1, & \text{falls es ein } s \in \mathbb{Z} \text{ mit } t <^{\mathbb{Z}} s \text{ und } f(s) = 1 \text{ gibt,} \\ 0, & \text{sonst} \end{cases}$$

(b) $G\varphi$ soll die Bedeutung von φ *wird immer der Fall sein* erhalten:

$$G^{\mathcal{D}}(f)(t) := \begin{cases} 1, & \text{falls } f(s) = 1 \text{ für alle } s \in \mathbb{Z} \text{ mit } t <^{\mathbb{Z}} s \text{ gilt,} \\ 0, & \text{sonst} \end{cases}$$

(c) $P\varphi$ soll die Bedeutung von φ *war einmal der Fall gewesen* erhalten:

$$P^{\mathcal{D}}(f)(t) := \begin{cases} 1, & \text{falls es ein } s \in \mathbb{Z} \text{ mit } s <^{\mathbb{Z}} t \text{ und } f(s) = 1 \text{ gibt,} \\ 0, & \text{sonst} \end{cases}$$

(d) $H\varphi$ soll die Bedeutung von φ *war immer der Fall gewesen* erhalten:

$$H^{\mathcal{D}}(f)(t) := \begin{cases} 1, & \text{falls } f(s) = 1 \text{ für alle } s \in \mathbb{Z} \text{ mit } s <^{\mathbb{Z}} t \text{ gilt,} \\ 0, & \text{sonst} \end{cases}$$

(e) $S(\varphi, \psi)$ soll die Bedeutung von φ *war einmal der Fall gewesen, und seitdem gilt* ψ (engl. *since* φ, ψ) erhalten:

$$S^{\mathcal{D}}(f, g)(t) := \begin{cases} 1, & \text{falls } f(s) = 1 \text{ und } g(r) = 1 \text{ für} \\ & \text{ein } s <^{\mathbb{Z}} t \text{ und alle } r \text{ mit } s <^{\mathbb{Z}} r \leq^{Z} t, \\ 0, & \text{sonst} \end{cases}$$

(f) $U(\varphi, \psi)$ soll die Bedeutung von φ *wird einmal der Fall sein, und bis dann gilt* ψ (engl. *until* φ, ψ) erhalten:

$$S^{\mathcal{D}}(f, g)(t) := \begin{cases} 1, & \text{falls } f(s) = 1 \text{ und } g(r) = 1 \text{ für} \\ & \text{ein } s >^{\mathbb{Z}} t \text{ und alle } r \text{ mit } t \leq^{\mathbb{Z}} r <^{Z} s, \\ 0, & \text{sonst} \end{cases}$$

Damit wird die Auswertungsfunktion $[\![\cdot]\!]^{\mathcal{D}}_{\cdot}(\cdot)$ definiert, die einem Ausdruck α, einem Zeitpunkt t und einer Belegung $h : \mathit{Var} \to U$ einen Wert $[\![\alpha]\!]_t(h)$ zuordnet:

(a) $[\![x]\!]_t(h) := h(x)$,

(b) $[\![f(t_1, \ldots, t_n)]\!]^{\mathcal{D}}_t(h) := f^{\mathcal{D}'}([\![t_1]\!]_t(h), \ldots, [\![t_n]\!]_t(h))$,

(c) $[\![\rho(t_1, \ldots, t_n)]\!]_t(h) := \rho^{\mathcal{D}'}(t)([\![t_1]\!](h), \ldots, [\![t_n]\!](h))$,

(d) $[\![(\varphi_1 \wedge \varphi_2)]\!]^{\mathcal{D}}_t(h) := \wedge^{\mathbb{B}}([\![\varphi_1]\!]_t(h), [\![\varphi_2]\!]_t(h))$,

(e) $[\![Qx\varphi]\!]_t(h) := Q^{\mathcal{D}'}(a \mapsto [\![\varphi]\!]_t(h[x/a]))$, für Quantoren Q,

(f) $[\![O\varphi]\!]_t(h) := O^{\mathcal{D}}(t' \mapsto [\![\varphi]\!]_{t'}(h))(t)$, für einstellige und

$$[\![O(\varphi_1, \varphi_2)]\!]_t(h) := O^{\mathcal{D}}(t' \mapsto [\![\varphi_1]\!]_{t'}(h),\ t' \mapsto [\![\varphi_2]\!]_{t'}(h))(t)$$

für zweistellige Temporaloperatoren O.

An der Definition sieht man, daß für jeden Term s der Wert $[\![s]\!]^{\mathcal{D}}_t(h)$ derselbe ist wie in der Quantorenlogik, nämlich $[\![s]\!]^{\mathcal{D}'}(h)$, also unabhängig vom Zeitpunkt t. Dagegen sind hängen die Wahrheitswerte $[\![\varphi]\!]_t(h)$ von Formeln φ nicht nur von den Extensionen $\rho^{\mathcal{D}'}(t)$ der darin auftretenden Prädikate ρ zum Zeitpunt t ab, sondern i.a. auch von den Extensionen $\rho^{\mathcal{D}'}(s)$ zu anderen Zeitpunkten s.

Man beachte, daß es hier keine Objekte zur Repräsentation von Ereignissen gibt. Die Temporaloperatoren vergleichen sozusagen die Zeiträume

$$\{t \in \mathbb{Z} \mid [\![\varphi]\!]_t = 1\},$$

die die durch Aussagen φ beschriebenen Ereignisse einnehmen. (Man könnte auch sagen: die einzigen Ereignisse sind, daß Formeln wahr bzw. falsch werden.)

5.1.3 Eine natürliche Sprache mit Temporaladverbien: TNL

In natürlichen Sprachen wird die zeitliche Anordnung, die zwischen einem erwähnten Vorgäng oder Ereigniss und der Äußerung besteht, oft nur durch das syntaktische Tempus von Sätzen ausgedrückt, d.h. morphologisch durch Unterschiede in den Verbformen oder syntaktisch durch die Verwendung von bestimmten Hilfsverben. Die Bedeutung der Äußerung hängt zwar wesentlich vom Äußerungszeitpunkt ab, aber dieser wird dann nicht ausdrücklich genannt.

Den Wert $[\![\alpha]\!]$ eines Ausdrucks α müssen wir daher von einem *impliziten* Äußerungszeitpunkt t abhängig machen, also eine Auswertung $[\![\alpha]\!]_t$ definieren. (Da t nicht nur zur Interpretation des optionalen Adverbs *jetzt* dient, das man als freie Zeitvariable auffassen könnte, kann man $[\![\alpha]\!]_t$ nicht als $[\![\alpha]\!]_{[jetzt/t]}$ verstehen.)

Damit die Syntax der Beispielsprache TNL einfach bleibt, ignorieren wir das syntaktische Tempus und benutzen weiterhin nur die Präsensform des Verbs. Sie bezieht sich auf den impliziten Äußerungszeitpunkt t. Explizite Angaben über Zeitpunkte werden in TNL nur durch Temporaladverbien gemacht. Nomina und Eigennamen sollen wie bisher unabhängig von der Zeit, die Verben aber von der Zeit abhängig interpretiert werden.

Wir erweitern die Sprache QNL um eine Kategorien der temporalen Adverbiale, wobei wir nach ihrer Funktionsweise drei Arten unterscheiden: *definite Zeitadverbiale ADVtd*, die einen bestimmten, vom Äußerungszeitpunkt unabhängigen Zeitpunkt angeben, *indexikalische Zeitadverbiale ADVti*, die einen bestimmten, vom Äußerungszeitpunkt abhängigen Zeitpunkt angeben, und *quantifizierende Zeitadverbien* oder temporale Satzjunktoren, die über Zeitpunkte quantifizieren.

Die Temporaladverbien werden nur am Satzende erlaubt, da sonst weitere Satzregeln mit anderen Verbstellungen nötig würden.

Beispiel 5.2 Die Änderungen gegenüber QNL (vgl. Beispiel 4.7) sind:

Syntaktische Kategorie	Metavariable und Konstruktionsmöglichkeiten	
definite Temp.Adverbial	*ADVtd* ::=	*an Christi Geburt*
	|	*zur Jahrtausendwende* | ...
indexik. Temp.Adverbial	*ADVti* ::=	*heute* | *morgen* | *gestern*
	|	*vor zehn Jahren* | *in drei Wochen*
Temporaloperator	*ADVtq* ::=	*immer*$^{(1)}$ | *irgendwann*$^{(1)}$ | *nie*$^{(1)}$
	|	*bis*$^{(2)}$ | *seit*$^{(2)}$
Satz *S*	φ ::=	... | φ *ADVti* | φ_1 *ADVtq*$^{(2)}$ φ_2

5.1.4 Interpretation von TNL

Eine Interpretation für TNL,

$$\mathcal{D} = (\mathcal{T}, \mathcal{D}', D_{ADVti}, D_{ADVtq}, c_1^{\mathcal{D}}, \ldots, d_1^{\mathcal{D}}, \ldots, q_1^{\mathcal{D}}, \ldots)$$

besteht aus einer Zeitstruktur

$$\mathcal{T} = (\mathbb{Z}, <^{\mathbb{Z}}, +^{\mathbb{Z}}, -^{\mathbb{Z}}, c_1^{\mathbb{Z}}, \ldots),$$

dem Bereich $D_{ADVtd} = \mathbb{Z}$ der Zeitpunkte und Bedeutungen für atomare definite Zeitadverbiale c_i und geeigneten Hilfsfunktionen wie $+^{\mathbb{Z}}$, und einer Interpretation

$$\mathcal{D}'$$

der anderen Ausdrücke wie bei QNL, so daß:

(a) Werte der definiten Zeitadverbiale c_i sind *Zeitpunkte*:

$$D_{ADVtd} = \mathbb{Z}, \tag{26}$$

darunter –wenn wir die Ausdrücke hier einmal als atomare verstehen–

$$\textit{an Christi Geburt}^{\mathcal{D}} := c_1^{\mathbb{Z}}, \quad \textit{zur Jahrtausendwende}^{\mathcal{D}} := c_2^{\mathbb{Z}}, \quad \ldots$$

(b) Werte der indexikalischen Zeitadverbiale d_i sind *zeitabhängige Zeitpunkte*:

$$D_{ADVti} = \mathbb{Z} \to \mathbb{Z}, \tag{27}$$

darunter

$$\textit{gestern}^{\mathcal{D}}(t) := t - 1, \quad \textit{heute}^{\mathcal{D}}(t) = t, \quad \textit{morgen}^{\mathcal{D}}(t) = t + 1,$$

(c) Werte der quantifizierenden Zeitadverbiale q_i sind *Zeitquantoren*, d.h. Tests auf (Tupel von) Test auf Zeitpunkten:

$$D_{ADVtq} = (\mathbb{Z} \to \mathbb{B})^n \to \mathbb{B} \tag{28}$$

darunter

$$\begin{array}{rcl} \textit{immer}^{\mathcal{D}}(f) & := & \begin{cases} 1, & \text{falls } f(t) = 1 \text{ für alle } t \in \mathbb{Z}, \\ 0, & \text{sonst} \end{cases} \\ \textit{irgendwann}^{\mathcal{D}}(f) & := & \begin{cases} 1, & \text{falls } f(t) = 1 \text{ für mindestens ein } t \in \mathbb{Z}, \\ 0, & \text{sonst} \end{cases} \\ \textit{nie}^{\mathcal{D}}(f) & := & \begin{cases} 1, & \text{falls } f(t) = 1 \text{ für kein } t \in \mathbb{Z}, \\ 0, & \text{sonst} \end{cases} \end{array}$$

und die zweistelligen Zeitquantoren $\textit{seit}^{\mathcal{D}} := S^{\mathcal{D}}$ und $\textit{bis}^{\mathcal{D}} := U^{\mathcal{D}}$ aus der Temporallogik TL (vgl. Beispiel 5.1).

Allerdings muß die Bedeutung $\rho^{\mathcal{D}'}$ der Prädikate ρ so geändert werden, daß der Wahrheitswert von Aussagen zu verschiedenen Zeitpunkten verschieden sein kann. Daher brauchen wir für die Prädikate eine andere Bedeutung:

(a) Werte der der Prädikate sind Funktionen, die jedem Zeitpunkt einen Test auf Individuen (d.h. eine Bedeutung von Prädikaten im bisherigen Sinn) zuordnen:

$$D_{V^{(n)}} = \{f \mid f : \mathbb{Z} \to (U^n \to \mathbb{B})\} \tag{29}$$

Für die Auswertungsfunktion beschränken wir uns auf Aussagen über Zustände, die *zu einem bestimmten Zeitpunkt* gelten, und ignorieren die Ereignisse, weil diese nicht zu einem Zeitpunkt, sondern während eines Zeitintervalls stattfinden.

Dann sind die Werte von der Interpretation $\mathcal{D}$ und einem Zeitpunkt t abhängig:

(a) der Wert quantifizierter Nominalphrasen ist zeitunabhängig:

$$[\![\tilde{\tau}]\!]_t^{\mathcal{D}} = [\![\tilde{\tau}]\!]^{\mathcal{D}'},$$

dasselbe gilt für Artikel, Nomina, Junktoren, Individuenquantoren,

(b) der Wert indexikalischer Zeitadverbien ist ein vom Auswertungszeitpunkt t abhängender Zeitpunkt:

$$[\![ADVti]\!]_t := ADVti^{\mathcal{D}}(t) \in \mathbb{Z},$$

beispielsweise $[\![morgen]\!]_t = t + 1$; analog müßte es für zusammengesetzte indexikalische Zeitadverbien wie *morgen in drei Tagen* gemacht werden,

(c) der Wert von Sätzen mit einem indexikalischen Zeitadverbial ist der Wert bei dem durch das Adverbial gegebenen Zeitpunkt:

$$[\![S\ ADVti]\!]_t := [\![S]\!]_{([\![ADVti]\!]_t)},$$

beispielsweise $[\![\textit{Emil kommt in drei Tagen}]\!]_t = [\![\textit{Emil kommt}]\!]_{t+3T}$

(d) der Wert von Sätzen mit quantifizierendem einstelligen Zeitadverbial, etwa *irgendwann*, wird durch Anwenden des entsprechenden Zeitquantors auf die durch den eingebetteten Satz definierte Funktion $f : \mathbb{Z} \to \mathbb{B}$ ermittelt:

$$[\![S\ ADVtq]\!]_t := [\![ADVtq]\!](s \mapsto [\![S]\!]_s)(t),$$

wobei der Zeitquantor auch den Äußerungszeitpunkt t berücksichtigen muß, wie in *irgendwann später (als jetzt)*,

(e) der Wert von Sätzen mit quantifizierendem zweistelligen Zeitadverbial, etwa *seit*, wird durch Anwenden des zweistelligen Zeitquantors auf die durch die eingebetteten Sätze definierten Funktionen aus $\mathbb{Z} \to \mathbb{B}$ berechnet:

$$[\![S_1\ ADVtq\ S_2]\!]_t := [\![ADVtq]\!](s \mapsto [\![S_1]\!]_s,\ s \mapsto [\![S_2]\!]_s)(t).$$

Auch hier muß der Äußerungszeitpunkt berücksichtigt werden.

Nicht behandelt werden:

(a) Ausdrücke zur Bezeichnung der Zeitdauer: ⟨*Zahlwort*⟩⟨*Einheit*⟩

(b) Ausdrücke zur *Häufigkeit* von Ereignissen: (absolut, nicht vergleichbar) *nie*, *manchmal*, *ständig*, *immer*

Häufigkeit = Vorkommen pro (Vergleichs)Zeitraum: (relativ, vergleichbar) *selten*, *oft*, und die damit gebildeten Vergleiche: *seltener als*, *(ungefähr,genau) so oft, wie*, *öfter als*

Bei der Implementierung einer Auswertung von Ausdrücken, die Zeitadverbiale enthalten, führt die Annahme *unendlich vieler Zeitpunkte* zu einer Schwierigkeit: Ausdrücke wie „φ *gilt immer*“ erfordern, daß man $[\![\varphi]\!]_t$ für *alle* $t \in \mathbb{Z}$ bestimmt, was unmöglich ist. Man sollte daher die Zeit durch endlich viele Objekte repräsentieren, z.B. durch die „benannten“ Zeitpunkte.

Bemerkung 5.3 Wenn man das syntaktische Tempus ebenfalls behandeln will, muß man beachten, daß ein indexikalisches Temporaladverb (des Matrixsatzes) sich auf den Äußerungszeitpunkt bezieht, nicht auf den der beschriebenen Situation: *Emil arbeitete gestern* bedeutet: *Gestern galt: Emil arbeitet*, und nicht: *Gestern galt: Emil arbeitete*. Also muß man, wenn man das Temporaladverb aus dem Satz extrahiert und als Temporaloperator behandelt, im entstehenden eingebetteten Satz das syntaktische Tempus korrigieren.

5.2 Exkurs: Programmierung der Interpretation von TNL

Wir wollen die frühere Interpretation `D` so (im Sinne der Kategorien des Aristoteles) ändern, daß die Bedeutung der Verben zeitabhängig wird, aber die Bedeutung der Eigennamen und Nomina zeitunabhängig bleibt.

5.2.1 Zeitpunkte und Temporaloperatoren

Als Zeitpunkte erlauben wir ganze Zahlen. Zeitadverbien können durch (einfache oder verallgemeinerte) Quantifizierungen über Zeitpunkte interpretiert werden. Damit diese bei der Auswertung terminieren, lassen wie die Quantoren nur über eine endliche Liste `[0,...,49]` laufen.

Die Bedeutung der Zeitadverbien wird dann mit Hilfe der auf die Zeitpunkte bezogenen Quantoren definiert.

⟨*TNL/zeit.sml*⟩≡

```
structure Zeit =
  struct
     type zeit = int
     val zeitpunkte = (* Zeitpunkte [0,1,...,49] *)
         List.tabulate (50, fn x => x)

     structure Q = Quantoren(struct type obj = zeit
                                    val objects = zeitpunkte
                             end)

     (* Temporaloperatoren als Quantoren ueber Zeitpunkte *)

     fun F S t = Q.exists (fn s => s > t) S   (* wird einmal *)
     fun G S t = Q.forall (fn s => s > t) S   (* wird immer *)

     fun H S t = Q.exists (fn s => s < t) S   (* war einmal *)
     fun P S t = Q.forall (fn s => s < t) S   (* war immer *)

     fun Since (S1,S2) t =
         case (List.find (fn s => s < t andalso S1 s) zeitpunkte)
          of SOME s => List.all (fn r => not(s < r andalso r <= t)
                                          orelse S2 r) zeitpunkte
           | NONE => false

     fun Until (S1,S2) t =
         case (List.find (fn s => t < s andalso S1 s) zeitpunkte)
          of SOME s => List.all (fn r => not(t <= r andalso r < s)
                                          orelse S2 r) zeitpunkte
           | NONE => false
```

```
    (* Werte von Zeitadverbien *)

    fun gestern t = t-1
    fun heute t   = t
    fun morgen t  = t+1

    fun immer S t      = List.all S zeitpunkte
    fun irgendwann S t = List.exists S zeitpunkte
    fun nie S t        = not (irgendwann S t)

    (* Werte von temporalen Junktoren *)

    val seit = Since
    val bis = Until
  end
```

5.2.2 Objekte mit zeitabhängigen Beziehungen

Die Struktur `Personen` wird so geändert, daß die Bedeutung der Verben von einem weiteren Argument z für Zeitpunkte abhängt.

⟨*TNL/personen.sml*⟩≡

```
 structure Personen =
   struct
     datatype person = a | b | c | d | e | f | g | h | i | j | k
                     | l | m | n | p | q | r | s | t | u | v | w

     structure Q =
       Quantoren(struct type obj = person
                        val objects = [a,b,c,d,e,f,g,h,i,j,k,l,m,n,p,q,r,s,t]
                 end)

     (* Werte der verallgemeinerten Quantoren ueber Personen *)

     val ein   = Q.exists
     val jeder = Q.forall
     val der   = Q.thefirst

     val alle    = Q.forall_2_2
     val einige  = Q.exists_2_2
     val je_zwei = Q.forall_1_2
     val ein_paar = Q.exists_1_2

     (* Werte der einfachen zeitunabhaengigen Ausdruecke *)
```

```
val Anna = a
val Dieter = d
val Emil = e
val Maria = m

fun Frau(a | b | c | g | h | i | m | n | s | t | u) = true
  | Frau(_) = false                (* Kurzschreibweise *)

fun Mann(d | e | f | j | k | l | p | q | r | v | w) = true
  | Mann(_) = false

fun Student(u | v | w) = false
  | Student(_) = true

fun Bruder ((d,e) | (e,d)) = true
  | Bruder (_,_) = false

fun Nachbar ((a,d) | (a,e) | (b,a) | (b,c) | (d,e)) = true
  | Nachbar (_,_) = false

fun ist(x,y) = if x=y then true else false

(* Werte der einfachen zeitabhaengigen Ausdruecke *)

fun arbeitet(z)(a) = (5 < z andalso z <= 10)
  | arbeitet(z)(m) = (z = 9 orelse z = 11)
  | arbeitet(z)(c) = true
  | arbeitet(z)(d) = true
  | arbeitet(z)(e) = true
  | arbeitet(z)(_) = false

fun kennt(z)(a,e) = z <= 10
  | kennt(z)(a,m) = z = 10
  | kennt(z)(e,a) = z >= 10
  | kennt(z)(m,e) = true
  | kennt(z)(x,y) = if x=y then true else false

fun liebt(z)(e,m) = true
  | liebt(z)(a,e) = true
  | liebt(z)(d,e) = true
  | liebt(z)(e,d) = true
  | liebt(z)(_,_) = false

val toString = (fn a => "a" | b => "b" | c => "c"
                 | d => "d" | e => "e" | m => "m"
                   | _ => "some person")
```

```
  end
```

Einfachheitshalber verzichten wir darauf, die Bedeutung von Nomen wie *Student* oder *Nachbar* von der Zeit abhängig zu machen.

5.2.3 Interpretation von TNL

Die Interpretation für TNL wird nun aus den bisher definierten Strukturen `Boole`, `Personen` und `Zeit` zusammengesetzt. (Eigentlich haben wir eine Struktur mit drei Universen, aber nur für die der Personen und der Zeit gibt es Quantoren.)

⟨*TNL/tnl.sml*⟩≡

```
 structure D =
  struct
     structure B = Boole
     structure P = Personen
     structure Z = Zeit

     type zeit   = Zeit.zeit
     type person = Personen.person

     (* Wertebereiche der syntaktischen Kategorien: *)

     type PN = person
     type NP = person
     type N  = person -> bool
     type N2 = person * person -> bool
     type IV = zeit -> (person -> bool)
     type TV = zeit -> (person * person -> bool)
     type S  = zeit -> bool

     type ADVtq  = S -> S
     type ADVtq2 = S * S -> S
     type ADVti  = zeit -> zeit
  end
```

Bei den Wertebereichen der syntaktischen Kategorien ist zu beachten: Werte der intransitiven und transitiven Verben sind jetzt Funktionen der Zeit, Werte der Sätze sind zeitabhängige Wahrheitswerte, und Werte der quantifizierenden Zeitadverbien wandeln Werte von Sätzen in andere Werte von Sätzen um.

5.2.4 Grammatik und zeitabhängige Auswertung für TNL

Die Auswertungsfunktion bei den Grammatikregeln liefert bei Verben jetzt zeitabhängige Beziehungen zwischen Personen und bei Sätzen zeitabhängige

Wahrheitswerte.

Die Syntax von QNL wird um indexikalische und quantifizierende Zeitadverbien erweitert; definite Zeitadverbien lassen wir weg, um keine Präpositionalphrasen einführen zu müssen. Zeitadverbien dürfen nur am Satzende stehen, damit die Verbzweitstellung ausreicht.

⟨*TNL/tnl.grm*⟩≡

```
structure B = D.B
structure P = D.P
structure Z = D.Z

val jetzt = 10      (* Auswertungsindex t *)

datatype value = object of D.PN | truth of bool | time of Z.zeit
exception Undefined

%%
%header (functor NL_LrValsFun (structure Token : TOKEN) : NL__LRVALS)

%term
     EOF | PUNKT
   | UND | ODER | NICHT
   | DET
   | IST | SIND
   | pn of D.PN
   | n of D.N
   | n2 of D.N2
   | ivl of D.IV
   | tv of D.TV
   | JEDER | EIN | ALLE | EINIGE
   | refl
   | rezi
   | advti of D.ADVti
   | advtq of D.ADVtq
   | advtq2 of D.ADVtq2

%nonterm
   start of (value * string) option
   | satz of D.S
   | np of D.NP
   | iv of D.IV
   | stv of D.TV
   | qnp of D.N -> bool
   | qnp2 of D.N2 -> bool

%start start
```

```
%eop EOF PUNKT
%noshift EOF PUNKT
%pos int

%left UND
%left ODER

%name NL_
%verbose

%%

start : satz              (SOME (truth(satz jetzt),
                                 Bool.toString (satz jetzt)))
     | np                 (SOME (object(np), P.toString np))
     | advti              (SOME (time(advti jetzt),
                                 Int.toString (advti jetzt)))
     |                    (NONE)

satz : np IST np          (fn t => P.ist(np1,np2))
     | np IST NICHT np    (fn t => B.nicht(np1 = np2))
     | np iv              (fn t => iv(t)(np))
     | np iv NICHT        (fn t => B.nicht(iv(t)(np)))
     | qnp iv             (fn t => qnp(fn x => iv(t)(x)))
     | NICHT qnp iv       (fn t => B.nicht(qnp(fn x => iv(t)(x))))
     | np tv np           (fn t => tv(t)(np1,np2))
     | np tv np NICHT     (fn t => B.nicht(tv(t)(np1,np2)))
     | qnp tv np          (fn t => qnp(fn x => tv(t)(x,np)))
     | NICHT qnp tv np    (fn t => B.nicht(qnp(fn x => tv(t)(x,np))))
     | satz UND satz      (fn t => B.und (satz1 t,satz2 t))
     | satz ODER satz     (fn t => B.oder(satz1 t,satz2 t))
     | qnp2 stv           (fn t => (qnp2 (stv t)))
     | qnp2 SIND n2       (fn t => (qnp2 n2))
     | satz advti         (fn t => satz (advti t))
     | satz advtq         (advtq satz)
     | satz advtq2 satz   (advtq2 (satz2,satz1))

np : pn                   (pn)
     | DET n              (P.der n)

qnp: JEDER n              (P.jeder n)
     | EIN n              (P.ein n)

qnp2 : ALLE n2            (P.alle n2)
     | EINIGE n2          (P.einige n2)
```

```
     | ALLE n             (P.je_zwei n)
     | EINIGE n           (P.ein_paar n)

iv : ivl                  (ivl)
     | tv refl            (fn t => (fn x => tv(t)(x,x)))

stv : tv rezi             (fn t => (fn (x,y) =>
                                       B.und(tv(t)(x,y),tv(t)(y,x))))
     | SIND n             (fn t => (fn (x,y) => B.und(n(x),n(y))))
```

Nach den Auswertungsregeln haben die Zeitquantoren weiten Wirkungsbereich. Daher hat der Satz *Alle Studenten arbeiten irgendwann* die unerwünschte Lesart *Es gibt einen Zeitpunkt, an dem alle Studenten arbeiten.* Vielleicht wäre es also besser, Temporaladverbien als Verbmodifikatoren zu interpretieren.

5.2.5 Erweiterung des Lexers

Zeitadverbien waren in der Eingabe bisher nicht erlaubt. Damit sie erkannt werden, muß der Lexer erweitert werden:

⟨*TNL/tnl.lex*⟩≡

```
structure B = D.B
structure P = D.P
structure Z = D.Z

structure Tokens = Tokens
structure Interface = Interface
open Interface

type pos = Interface.pos
type svalue = Tokens.svalue
type ('a,'b) token = ('a,'b) Tokens.token
type lexresult = (svalue,pos) token

fun eof() = Tokens.EOF(!line,!line)
fun makeInt (s : string) = s

%%
%header (functor NL_LexFun(structure Tokens: NL__TOKENS
                           structure Interface: INTERFACE) : LEXER);
%full

idchars=[0-9]|[_&+/@|#*`'><];
alpha=[a-zA-ZÄÖÜäöüß];
comment=%(.*)\n;
whitespace=[\t\ ]*;
```

```
%%
<INITIAL>{whitespace} => (lex());
<INITIAL>"Der"    => (Tokens.DET(!line,!line));
<INITIAL>"Die"    => (Tokens.DET(!line,!line));
<INITIAL>"Jede"   => (Tokens.JEDER(!line,!line));
<INITIAL>"Jeder"  => (Tokens.JEDER(!line,!line));
{comment}  => (next_line(); lex());
\n         => (next_line(); lex());
"."        => (Tokens.PUNKT(!line,!line));
"und"      => (Tokens.UND(!line,!line));
"oder"     => (Tokens.ODER(!line,!line));
"nicht"    => (Tokens.NICHT(!line,!line));
"der"      => (Tokens.DET(!line,!line));
"die"      => (Tokens.DET(!line,!line));
"ist"      => (Tokens.IST(!line,!line));
"Anna"     => (Tokens.pn(P.Anna,!line,!line));
"Dieter"   => (Tokens.pn(P.Dieter,!line,!line));
"Emil"     => (Tokens.pn(P.Emil,!line,!line));
"Maria"    => (Tokens.pn(P.Maria,!line,!line));
"Mann"     => (Tokens.n(P.Mann,!line,!line));
"Frau"     => (Tokens.n(P.Frau,!line,!line));
"Student"  => (Tokens.n(P.Student,!line,!line));
"arbeitet" => (Tokens.ivl(P.arbeitet,!line,!line));
"kennt"    => (Tokens.tv(P.kennt,!line,!line));
"liebt"    => (Tokens.tv(P.liebt,!line,!line));
"jede"     => (Tokens.JEDER(!line,!line));
"jeder"    => (Tokens.JEDER(!line,!line));
"eine"     => (Tokens.EIN(!line,!line));
"ein"      => (Tokens.EIN(!line,!line));
"alle"     => (Tokens.ALLE(!line,!line));
"einige"   => (Tokens.EINIGE(!line,!line));
"sich"     => (Tokens.refl(!line,!line));
"einander" => (Tokens.rezi(!line,!line));
"Nachbarn" => (Tokens.n2(P.Nachbar,!line,!line));
"Brüder"   => (Tokens.n2(P.Bruder,!line,!line));
"Männer"   => (Tokens.n(P.Mann,!line,!line));
"Frauen"   => (Tokens.n(P.Frau,!line,!line));
"Studenten"  => (Tokens.n(P.Student,!line,!line));
"kennen"   => (Tokens.tv(P.kennt,!line,!line));
"lieben"   => (Tokens.tv(P.liebt,!line,!line));
"sind"     => (Tokens.SIND(!line,!line));
"gestern"  => (Tokens.advti(Z.gestern,!line,!line));
"heute"    => (Tokens.advti(Z.heute,!line,!line));
"morgen"   => (Tokens.advti(Z.morgen,!line,!line));
"immer"    => (Tokens.advtq(Z.immer,!line,!line));
"nie"      => (Tokens.advtq(Z.nie,!line,!line));
```

```
"irgendwann" => (Tokens.advtq(Z.irgendwann,!line,!line));
"bis"        => (Tokens.advtq2(Z.bis,!line,!line));
"seit"       => (Tokens.advtq2(Z.seit,!line,!line));
.            => (error ("ignoring illegal character " ^ yytext,
                          !line,!line); lex());
```

5.2.6 Automatische Erstellung des Programmpakets für TNL

Im `%header` von `tnl.grm` und `tnl.lex` wurden die Namen der Funktoren wieder genau so gelassen, wie sie für `NL` waren, damit das Interface und der Linker einfach übernommen werden können.

Um das Paket benutzen zu können, muß man analog zur Erstellung von QNL

(a) das Archiv `TNL.tgz` mit `> tar -xvfz TNL.tgz` auspacken,

(b) mit `> cd TNL` in das Verzeichnis `TNL` gehen,

(c) dort `> sml` (bzw. `sml-110.0.42`) starten,

(d) darin mit `- CM.make "tnl.110.42.cm";` die Struktur `NL` erzeugen,

(e) anschließend mit `- NL.parse();` den Parser für TNL-Sätze starten.

Die Beschreibungsdatei, die der Compilation Manager dazu braucht, ist (mit Anpassung für die LMU im Kommentar):

⟨*TNL/tnl.110.42.cm*⟩≡

```
Library
  structure Zeit
  structure Boole
  structure Personen
  structure NL
is
  $/smlnj-lib.cm
  $/ml-yacc-lib.cm
  $/basis.cm

  boole.sml
  quantoren.sml
  zeit.sml
  personen.sml
  tnl.sml
  tnl.lex          (* fuer Tokenizer *)
  tnl.grm          (* fuer Parser *)

  nl.interface.sml (* fuer Lexer und Linker *)
  nl.link.sml  (* an LMU: nl.link.110.0.3.sml *)
```

5.3 Adverbiale Bestimmungen des Orts

Die Behandlung von Ort und Raum ist komplizierter als die der Zeit, da der Raum 2- oder 3-dimensional ist und mehr Grundrelationen zwischen Stellen oder Gebieten im Raum sprachlich repräsentierbar sind.

Außerdem spielt der Ort von Gegenständen *relativ zur Stelle des Sprechers oder Hörers* im Raum ein wichtige Rolle, die in Präpositionen wie *vor, hinter, neben, über, unter* zur Sprache kommt, während es bei der Zeit eine von Sprecher und Hörer unabhängige lineare Anordnung gibt.

Wir betrachten nur einen zweidimensionalen Raum, eine Ebene. Ähnlich wie bei der Zeit beobachtet man bestimmte Darstellungen des Raums in den natürlichsprachlichen Ausdrucksweisen:

(a) Raumpunkte oder -gebiete werden durch körperliche, sinnlich wahrnehmbare Gegenstände bestimmt und unterscheidbar, den Orten, an denen sich ein Gegenstand bzw. verschiedene Gegenstände befinden, etwa *dort, wo die drei Eichen stehen.* Sie können aber auch durch Ereignisse, die an diesem Ort stattfinden, bezeichnet werden, etwa *dort, wo der Pfeffer wächst.*

    Viele Orte haben einen Namen, der meist von unbeweglichen „Gegenständen“ wie landschaftlichen Besonderheiten herkommt.

(b) Räumliche Gebiete haben eine meßbare Ausdehnung, die durch Vielfache eines Einheitsgebiets angegeben wird, z.B. der einer landwirtschaftlichen Fläche. Die Ausdehnungen verschiedener Gebiete lassen sich mit einander vergleichen, indem man diese Vielfachen im Zahlbereich vergleicht.[25]

(c) Je zwei Orte eines Raums, sozusagen punktförmige Gebiete der Ausdehnung 0, haben einen *Abstand zu* oder eine *Entfernung von* einander, die in Vielfachen einer Einheitsentfernung gemessen werden.

(d) Zwei Orte eines beschränkten Gebietes haben, relativ zu einem Beobachtungsstandpunkt (außerhalb des Gebiets?), eine *Lage* zu einander: sie liegen *vor*, *hinter*, *neben* oder *bei* einander, im dreidimensionalen auch *unter* oder *über* einander.

(e) Gegenstände *befinden sich* zu bestimmten Zeitpunkten an bestimmten Orten, Ereignisse *finden* an Orten oder in Gebieten *statt*, und Zustände *herrschen* dort. Während sich nie zwei Gegenstände am selben Ort befinden, können zwei Ereignisse am selben Ort stattfinden.

(f) Über Orte kann quantifiziert werden: *überall*, *irgendwo*, *vielerorts*, *nirgends*.

[25]Eine Implementierung sollte natürlich die Vergleiche in allen Dimensionen wie Raum, Gewicht, Temperatur usw., die miteinander verglichen werden, indem man Vielfache eines Einheitswerts als Zahlen vergleicht, auf diese Zahlvergleiche reduzieren.

Wissenschaftliche Aussagen, insbesondere die der Mathematik, sind orts- und zeitunabhängig: ihre Gültigkeit hängt nicht vom Ort oder Zeitpunkt ihrer Äußerung ab. Die Gültigkeit umgangssprachlicher Aussagen hängt dagegen normalerweise von der Äußerungssituation ab.

Ausdrücke zur Bezeichnung von Orten werden verwendet, um Gegenstände durch den Ort, an dem sie sich befinden, Ereignisse durch den Ort, an dem sie stattfinden, oder Zustände durch den Ort, an dem sie herrschen, von anderen, gleichartigen Gegenständen, Ereignissen oder Zuständen zu unterscheiden, oder um die Gültigkeit von Aussagen auf einen Ort zu relativieren.

Wir wollen nun bei der Auswertung von Sätzen nur eine einfache Verwendung von Ortsangaben in Sätzen berücksichtigen:

(a) es werden nur Ortsangaben zugelassen, die sich durch eine lokale Präposition auf die Positionen von Gegenständen zurückführen lassen,

(b) die Ortsabhängigkeit von Aussagen soll nur darin bestehen, daß eine räumliche Beziehung zwischen der Position des Subjekts und dem durch das Ortsadverbial bezeichneten Ort oder Gebiet besteht.[26]

Durch diese Einschränkung ersparen wir uns, *Ereignisse* oder Zustände einzuführen. Die hier erlaubte Ortsabhängigkeit des Wahrheitswerts einer Aussage ist natürlich nur in bestimmten Fällen angemessen.

Beziehungen zwischen Zeit- und Raumangaben sollen aber berücksichtigt werden, z.B. die Wirkungsbereiche der Zeit- und Ortsquantoren im unbestimmten Futur: *Ich werde dort schlafen* entspricht dem Satz *Es wird einmal der Fall sein, daß ich dort bin und schlafe*, und nicht der Konjunktion *Ich werde einmal dort sein und ich werde einmal schlafen.*[27].

Wenn die lokalen Präpositionen durch Beziehungen zwischen Orten interpretiert werden sollen und Orte nur als Positionen von Individuen gegeben sind, sollte die Auswertung etwa

$$\begin{aligned} & [\![np_1 \ \textit{befindet sich neben}\ np_2]\!]_{(t,p)} \\ & \quad = \ [\![neben]\!](position([\![np_1]\!])(t), position([\![np_2]\!])(t), p) \end{aligned}$$

ergeben, wenn t der Zeitpunkt und p der Ort der Äußerung ist. Aber wie soll ein Ortsadverbial allein, ohne das Subjekt des Satzes, ausgewertet werden?

Der durch ein Ortsadverbial bestimmte Ort hängt im allgemeinen von der Zeit ab, z.B. hängt der durch *neben dem blauen Fahrzeug* bestimmte Ort davon ab, wo sich das Fahrzeug zum Bezugszeitpunkt befindet. Nimmt man als Bezugszeitpunkt vereinfachend den Äußerungszeitpunkt an, so braucht man:

$$[\![neben\ np_2]\!]_{(t,p)} = \lambda p_1.[\![neben]\!](p_1, position([\![np_2]\!])(t), p).$$

[26]Bei quantifiziertem Subjekt oder Adverbial sind die Positionen der individuellen Subjekte oder Orte entsprechender Basisaussagen gemeint.

[27]Der Satz *Ich schlafe dort* läßt keine Präsens-Lesart zu

Hierdurch wird der Wert des Ortsadverbials ein situations- und ortsabhängiger Wahrheitswert.

Für definite Ortsadverbiale vereinfachen wir das so, daß die Lokalpräpositionen nicht vom Ort der Äußerung abhängig sind. Die Argumente sollen nacheinander übergeben werden, und zwar zuerst die Position der eingebetteten Nominalphrase:

$$\begin{aligned}
[\![(neben\ np_2)]\!]_{(t,p)} &= \lambda p_1.[\![neben]\!](position([\![np_2]\!])(t))(p_1) \\
&= [\![neben]\!](position([\![np_2]\!])(t)).
\end{aligned}$$

Auf diese Weise kann die Bedeutung kategorematisch definiert werden. Die Lokalpräposition *neben* hat dann den Typ

$$[\![neben]\!] : Ort \to (Ort \to \mathbb{B}).$$

5.3.1 Eine natürliche Sprache mit Ortsadverbien: LNL

Wir erweitern die Sprache TNL um eine Kategorie der lokalen Adverbiale, wobei wir nach ihrer Funktionsweise drei Arten unterscheiden: *definite Lokaladverbiale ADVld*, die einen bestimmten, vom Äußerungsort unabhängigen, allerdings noch vom Äußerungszeitpunkt abhängigen Ort angeben, *indexikalische Lokaladverbiale ADVli*, die einen bestimmten, vom Äußerungsort abhängigen Ort angeben, und *quantifizierende Lokaladverbien* oder lokale Satzjunktoren, die über Orte quantifizieren.

Die Lokaladverbien werden nur am Satzende erlaubt (vor oder hinter den Temporaladverbien?), da sonst weitere Satzregeln mit anderen Verbstellungen nötig würden.

Beispiel 5.4 Die Änderungen gegenüber TNL (vgl. Beispiel 5.2) sind:

Syntaktische Kategorie	Metavariable und Konstruktionsmöglichkeiten	
lokale Präposition	$Prepl$ ::=	$vor \mid hinter \mid neben \mid \ldots$
definites Lokaladverbial	$ADVld$ ::=	$Prepl\ NP$
indexik. Lokaladverbial	$ADVli$ ::=	$hier \mid dort$
Ortsquantoren	$ADVlq$ ::=	$überall \mid irgendwo \mid nirgendwo$
Satz	φ ::=	$\ldots \mid \varphi\ ADVld$
	$\mid$	$\varphi\ ADVli \mid \varphi\ ADVlq$

Die Satzregeln erlauben zwar eine iterierte Verwendung von Lokaladverbien, aber daran ist hier eigentlich nicht gedacht.

5.3.2 Interpretation von LNL

Eine *Interpretation* für LNL,

$$\mathcal{D} = (\mathcal{T}, \mathcal{O}, \mathcal{D}', D_{ADVli}, D_{ADVlq}, c_1^{\mathcal{D}}, \ldots, d_1^{\mathcal{D}}, \ldots, q_1^{\mathcal{D}}, \ldots),$$

besteht aus einer Interpretation $(\mathcal{T}, \mathcal{D}', \ldots)$ für TNL, erweitert um

(a) eine Raumstruktur

$$\mathcal{O} = (O, neben^{\mathcal{O}}, vor^{\mathcal{D}}, \ldots, dist^{\mathcal{O}})$$

aus einer Menge $O := \mathbb{Z} \times \mathbb{Z}$ von Orten, Relationen[28]

$$vor^{\mathcal{O}}, neben^{\mathcal{O}} : O \times O \to \mathbb{B}$$

usw. zwischen Orten, und einer Abstandsfunktion

$$dist^{\mathcal{O}} : O \times O \to \mathbb{R}$$

zwischen Orten (von der wir unten aber keinen Gebrauch machen),

(b) Bedeutungen für die neuen Konstanten, die nicht schon in $\mathcal{O}$ interpretiert sind, darunter

$$position^{\mathcal{D}} : U \to (\mathbb{Z} \to O),$$

die Objekten $x \in U$ zu Zeitpunkten $t \in \mathbb{Z}$ einen Ort $position(x)(t)$ zuweist,

sodaß gilt:

(a) Werte der indexikalischen Ortsadverbiale c_i sind *ortsabhängige Orte*

$$D_{ADVli} := O \to O, \tag{30}$$

darunter $hier^{\mathcal{D}}(p) = p$ (wobei p der Ort der Äußerungssituation ist),

(b) Werte der quantifizierenden Zeitadverbiale q_i sind (einstellige) *Ortsquantoren*, d.h. Tests auf Tests auf Orten:

$$D_{ADVlq} := (O \to \mathbb{B}) \to \mathbb{B}, \tag{31}$$

darunter (die üblichen Quantoren über den Bereich O)

$$\begin{aligned}
\textit{überall}^{\mathcal{D}}(f) &:= \begin{cases} 1, & \text{falls } f(p) = 1 \text{ für alle } p \in O, \\ 0, & \text{sonst} \end{cases} \\
\textit{irgendwo}^{\mathcal{D}}(f) &:= \begin{cases} 1, & \text{falls } f(p) = 1 \text{ für mindestens ein } p \in O, \\ 0, & \text{sonst} \end{cases} \\
\textit{nirgendwo}^{\mathcal{D}}(f) &:= \begin{cases} 1, & \text{falls } f(p) = 0 \text{ für alle } p \in O, \\ 0, & \text{sonst} \end{cases}
\end{aligned}$$

(c) Werte der definiten Ortsadverbiale d_i sind[29] *situationsabhängige Eigen-*

[28] Wie gesagt, sollten dies eigentlich *von einem Ort*, dem Standpunkt des Beobachters oder dem Ort der Äußerung, *abhängige* Relationen zwischen Orten sein. In der Implementierung unten haben Lokalpräpositionen den Typ $O \to (O \to \mathbb{B})$, damit die Adverbiale kategorematisch interpretiert werden können.

[29] für die oben erläuterte eingeschränkte Verwendung von lokalen Präpositionen

schaften von Orten (der Position des Subjekts des Satzes),

$$D_{ADVld} = Sit \to (O \to \mathbb{B}), \tag{32}$$

die eigentlich nur vom Zeitanteil der Äußerungssituation abhängen.

Die Auswertungsfunktion $[\![\cdot]\!]^{\mathcal{D}}_{\cdot}$ ordnet den Ausdrücken α einen von der Äußerungssituation $s \in Sit = Zeit \times Ort$ abhängenden Wert $[\![\alpha]\!]^{\mathcal{D}}_{s}$ zu. Da die Bedeutung der Verben und Nomen nicht verändert wurde, spielt der Ortsanteil p der Situation $s = (t, p)$ nur bei wenigen Ausdrücken eine Rolle:

(a) der Wert quantifizierter Nominalphrasen ist situationsunabhängig:

$$[\![\tilde{\tau}]\!]^{\mathcal{D}}_{(t,p)} = [\![\tilde{\tau}]\!]^{\mathcal{D}'},$$

dasselbe gilt für Artikel, Nomina, Junktoren, Individuenquantoren,

(b) der Wert indexikalischer Zeitadverbien ist ein vom Auswertungszeitpunkt t abhängender Zeitpunkt:

$$[\![ADVti]\!]_{(t,p)} := ADVti^{\mathcal{D}}(t) \in \mathbb{Z},$$

(c) der Wert von Sätzen mit einem indexikalischen Zeitadverbial ist der Wert bei dem durch das Adverbial gegebenen Zeitpunkt:

$$[\![S\ ADVti]\!]_{(t,p)} := [\![S]\!]_{([\![ADVti]\!]_{(t,p)},p)},$$

wobei sich der Ortsanteil der Auswertungssituation nicht ändert,

(d) der Wert von Sätzen mit quantifizierendem einstelligen (analog bei zweistelligem) Zeitadverbial ergibt sich durch Anwenden des entsprechenden Zeitquantors auf die durch den eingebetteten Satz definierte Funktion:

$$[\![S\ ADVtq]\!]_{(t,p)} := [\![ADVtq]\!](t' \mapsto [\![S]\!]_{(t',p)})(t),$$

wobei der Zeitquantor auch den Äußerungszeitpunkt t berücksichtigt;

(e) der Wert indexikalischer Ortsadverbien ist ein vom Auswertungsort p abhängender Ort:

$$[\![ADVli]\!]_{(t,p)} := ADVli^{\mathcal{D}}(p) \in O,$$

(f) der Wert von Sätzen mit einem indexikalischen Ortsadverbial ist der Wert bei dem durch das Adverbial gegebenen Ort:

$$[\![S\ ADVli]\!]_{(t,p)} := [\![S]\!]_{(t,[\![ADVli]\!]_{(t,p)})},$$

wobei sich der Zeitanteil der Auswertungssituation nicht ändert,

(g) der Wert von Sätzen mit quantifizierendem Ortsadverbial ergibt sich durch Anwenden des entsprechenden Ortsquantors auf die durch den eingebetteten Satz definierte Funktion:

$$[\![S\ ADVlq]\!]_{(t,p)} := [\![ADVlq]\!](p' \mapsto [\![S]\!]_{(t,p')})(p),$$

wobei der Ortsquantor i.a. auch den Äußerungsort p berücksichtigt;

(h) der Wert eines definiten Ortsadverbials ist[30] eine *Eigenschaften von Orten*,

$$[\![Prepl\ NP]\!]_{(t,p)} = \lambda p' \in O.\ Prepl^{\mathcal{O}}(p', position^{\mathcal{D}}([\![NP]\!])(t)),$$

die auf das dem Satzsubjekt entsprechende Individuum zutreffen soll.

Da wir definite Ortsadverbiale als Bedingungen an die Position des Individuums, das dem Satzsubjekt entspricht, interpretieren wollen, können wir sie nicht als Satzoperatoren behandeln, weil man dann am Kernsatz das Subjekt nicht ablesen könnte. Daher nehmen wir folgende Bedingung:

(i) der Wert eines Satzes mit einem definiten Ortsadverbial ist die Konjunktion der Werte des Satzes ohne das Ortsadverbial und der Aussage, die Position des Satzsubjekts habe die durch das Adverbial ausgedrückte Eigenschaft:

$$[\![NP\ IV\ ADVld]\!]_{(t,p)} = [\![NP\ IV]\!]_{(t,p)} \wedge^{\mathbb{B}} [\![ADVld]\!]_{(t,p)}(position^{\mathcal{D}}([\![NP]\!])(t)),$$

und analog für transitive Verben.

Damit sich die Ortsquantoren auch auswirken, muß bei den atomaren Aussagen eine Abhängigkeit vom Ort der Auswertung $s = (t,p)$ eingebaut werden. Das machen wir nur für einfache Aussagen mit intransitivem Verb, da von den beiden Verben der Beispielsprache, `kennen` und `arbeiten`, nur das intransitive eine lokale Modifikation erlaubt:

(j) der Wert eines atomaren Satzes mit intransitivem Verb hängt auch von der Position des Subjekts in der Auswertungssituation ab:

$$[\![NP\ IV]\!]_{(t,p)} = [\![IV]\!](t)([\![NP]\!]) \wedge^{\mathbb{B}} (p = position^{\mathcal{D}}([\![NP]\!])(t)),$$

Damit wird ein Satz wie *Maria arbeitet* gleichbedeutend mit *Maria arbeitet hier und jetzt.* Das ist sicher nicht allgemein sinnvoll.

Bemerkung 5.5 Hier sind noch keine temporalen Adverbiale und auch keine quantifizierten Nominalphrasen in den lokalen Adverbialen betrachtet!

Definite Ortsadverbiale beziehen sich i.a. nicht auf die Positionen der Objekte zum Zeitpunkt t der Äußerungssituation $s = (t,p)$, sondern auf deren Positionen

[30] für die oben erläuterte eingeschränkte Verwendung von lokalen Präpositionen

zu einem Zeitpunkt, der durch temporale Adverbien bestimmt sein kann: in *Gestern kam jemand leise in den Raum und stand plötzlich neben mir.* ist mit *neben mir* die Position gemeint, an der ich mich *gestern* (in der Situation, über die gesprochen wird!) befand.

Man muß gewisse Abhängigkeiten zwischen der Quantifizierung über die Zeit und den Ort beachten. Bei individuellem Subjekt bewirkt der Ortsquantor *überall* eine implizite Zeitquantifizierung durch *irgendwann*, da ein Individuum sich zu jedem Zeitpunkt nur an einem Ort befindet: *Emil arbeitet überall* heißt, daß er im Lauf der Zeit an jedem Ort arbeitet. Bei existentiell quantifiziertem Subjekt gilt diese Beschränkung nicht. *Überall arbeitet jemand* heißt, daß gleichzeitig an jedem Ort, allerdings von verschiedenen Personen, gearbeitet wird.

5.4 Exkurs: Programmierung der Interpretation von LNL

Wir erweitern die bisherige Sprache um indexikalische und quantifizierende Ortsadverbiale.

5.4.1 Orte, Ortsquantoren und lokale Präpositionen

Als Orte erlauben wir Paare (i, j) von ganzen Zahlen. Quantifizierende Ortsadverbien wie *überall*, *irgendwo*, *nirgendwo* (ungewöhnlich: *mancherorts*) interpretieren wir durch Quantifizierungen über die Orte (i, j) mit $0 \leq i, j < 20$.

Lokale Präpositionen wie *neben*, *vor*, *hinter* (und analog *über*, *unter* im dreidimensionalen Raum) werden durch Relationen zwischen Orten interpretiert.[31]

⟨*TNL/ort.sml*⟩≡

```
structure Ort =
  struct
     type ort = int * int
     val orte =
         let
             val L = List.tabulate (20,fn z => z)
             fun f x = List.map (fn y => (x,y)) L
         in
             List.foldr (op @) nil (List.map f L)
         end

     fun irgendwo (S: ort -> bool) p = List.exists S orte
     fun nirgendwo (S : ort -> bool) p = not(List.exists S orte)
     fun ueberall (S : ort -> bool) p = List.all S orte

     fun neben(r,s)(p,q)  = (q = s andalso (p < r orelse r < p))
     fun vor(r,s)(p,q)    = (p = r andalso q < s)
     fun hinter(r,s)(p,q) = (p = r andalso q > s)

     fun hier (p:ort) = p
  end
```

Der Wert indexikalischer Ortsadverbien wie *hier*, *dort* hängt vom Ort der Äußerung ab (und nicht vom Ort des Ereignisses, von dem gesprochen wird).

Mit den Präpositionen *neben*, *vor*, *hinter* werden eigentlich indexikalische Ortsadverbien aufgebaut, da die Bedeutung der Präpositionen relativ zum Standort des Sprechers zu definieren wäre; das machen wir einfachheitshalber aber nicht.

[31] Die Gesamtheit der Orte, über die die Quantoren laufen, wird hier als Liste angegeben, um die Verwendung von Vektor- oder Feldtypen zu vermeiden.

5.4.2 Interpretation der Sprache LNL

Wir interpretieren alle Aussagen relativ zu einer Situation (t, p), die durch einen Zeitpunkt t und einen Ort p festgelegt ist. Verben werden wie in `tnl.sml` durch zeitabhängige, aber ortsunabhängige Beziehungen zwischen Individuen interpretiert, Nomina durch zeit- und ortsunabhängige.

Ein quantifizierendes Adverbial wird als Quantor über die jeweilige Dimension der Situationen behandelt. Zur Auswertung von Aussagen mit Ortsadverbialen ordnen wir den Individuen eine zeitabhängige Position zu und vergleichen die Position des Satzsubjekts mit der Ortsangabe aus dem Adverbial.

⟨*TNL/lnl.sml*⟩≡
```
structure D =
  struct
     structure B = Boole
     structure P = Personen
     structure Z = Zeit
     structure O = Ort

     type zeit = Z.zeit
     type ort = O.ort
     type person = Personen.person

     type sit = zeit * ort

     (* Position von Individuen im Laufe der Zeit *)

     fun position (P.a) (t:zeit) = (5,5)
       | position (P.m) t = if t < 10 then (5,4) else (5,t)
       | position (P.e) t = if t = 5 then (t+1,5) else (t,t)
       | position (_:person) (t:zeit) =
         if t < 10 then (t,0) else (t,1)

     (* Wertebereiche der syntaktischen Kategorien: *)

     type PN = person
     type NP = person
     type S = sit -> bool
     type N = person -> bool
     type N2 = person * person -> bool
     type IV = zeit -> (person -> bool)
     type TV = zeit -> (person * person -> bool)
     type Pl = ort -> ort -> bool

     type ADVtq = (zeit -> bool) -> (zeit -> bool)
     type ADVtq2 = (zeit -> bool) * (zeit -> bool) -> (zeit -> bool)
```

```
    type ADVti = zeit -> zeit

    type ADVlq = (ort -> bool) -> (ort -> bool)
    type ADVld = zeit -> (ort -> bool)
    type ADVli = ort -> ort
  end
```

5.4.3 Grammatik und Auswertung für LNL-Ausdrücke

Die Syntax von TNL wird um definite, quantifizierende und indexikalische Ortsadverbien erweitert; dazu brauchen wir auch lokale Präpositionen. Lokaladverbien dürfen nur am Satzende stehen, damit die Verbzweitstellung ausreicht.

⟨*TNL/lnl.grm*⟩≡

```
structure B = D.B
structure P = D.P
structure Z = D.Z
structure O = D.O

val jetzt = 10      (* Auswertungszeit t *)
val hier  = (5,5)   (* Auswertungsort  p *)

datatype value = object of D.PN | truth of bool
               | time of Z.zeit | place of D.ort
exception Undefined

%%
%header (functor NL_LrValsFun (structure Token : TOKEN) : NL__LRVALS)

%term
     EOF | PUNKT
   | UND | ODER | NICHT
   | DET
   | IST | SIND
   | pn of D.PN
   | n of D.N
   | n2 of D.N2
   | ivl of D.IV  (* lexikalisch intransitive Verben *)
   | tv of D.TV
   | JEDER | EIN | ALLE | EINIGE
   | refl | rezi
   | advti of D.ADVti
   | advtq of D.ADVtq
   | advtq2 of D.ADVtq2
   | pl of D.Pl
   | advli of D.ADVli
```

```
   | advlq of D.ADVlq

%nonterm
   start of (value * string) option
   | satz of D.S
   | np of D.NP
   | qnp of D.N -> bool
   | qnp2 of D.N2 -> bool
   | iv of D.IV
   | stv of D.TV
   | advld of D.ADVld

%start start

%eop EOF PUNKT
%noshift EOF PUNKT
%pos int

%left UND
%left ODER

%name NL_
%verbose

%%

start : satz              (SOME (truth(satz (jetzt,hier)),
                                 Bool.toString (satz (jetzt,hier))))
     | np                 (SOME (object(np), P.toString np))
     | advti              (SOME (time(advti jetzt),
                                 Int.toString (advti jetzt)))
     | advli              (SOME (place(advli hier),
                                 "(" ^ (Int.toString (#1 (advli hier))) ^
                                 "," ^ (Int.toString (#2 (advli hier))) ^ ")"))
     |                    (NONE)

satz : np IST np          (fn (t,p) => P.ist(np1,np2))
     | np IST NICHT np    (fn (t,p) => B.nicht(P.ist(np1,np2)))
     | np iv              (fn (t,p) => B.und(p = D.position(np)(t),iv(t)(np)))
     | np iv NICHT        (fn (t,p) => B.nicht(B.und(p = D.position(np)(t),
                                                    iv(t)(np))) )
     | qnp iv             (fn (t,p) => qnp(fn x => iv(t)(x)))
     | NICHT qnp iv       (fn (t,p) => B.nicht(qnp(fn x => iv(t)(x))))
     | np tv np           (fn (t,p) => tv(t)(np1,np2))
     | np tv np NICHT     (fn (t,p) => B.nicht(tv(t)(np1,np2)))
     | qnp tv np          (fn (t,p) => qnp(fn x => tv(t)(x,np)))
```

```
      | NICHT qnp tv np    (fn (t,p) => B.nicht(qnp(fn x => tv(t)(x,np))))
      | satz UND satz      (fn (t,p) => B.und (satz1 (t,p),satz2 (t,p)))
      | satz ODER satz     (fn (t,p) => B.oder(satz1 (t,p),satz2 (t,p)))
      | qnp2 iv            (fn (t,p) => qnp2 (fn (x,y) => B.und(iv(t)(x),
                                                             iv(t)(y))))
      | qnp2 stv           (fn (t,p) => qnp2 (stv t))
      | qnp2 SIND n2       (fn (t,p) => qnp2 n2)
      | satz advtq         (fn (t,p) => advtq(fn t' => satz (t',p))(t))
      | satz advtq2 satz   (fn (t,p) => advtq2(fn t' => satz2(t',p),
                                               fn t' => satz1(t',p))(t))
      | satz advti         (fn (t,p) => satz (advti t,p))
      | satz advlq         (fn (t,p) => advlq(fn p' => satz (t,p'))(p))
      | np iv advld        (fn (t,p) => B.und(iv(t)(np),
                                              advld(t)(D.position(np)(t))))
      | np iv advli        (fn (t,p) => B.und(iv(t)(np),
                                              p = advli(D.position(np)(t))))

 np : pn                   (pn)
      | DET n              (P.der n)

 qnp: JEDER n              (P.jeder n)
      | EIN n              (P.ein n)

 qnp2 : ALLE n2            (P.alle n2)
      | EINIGE n2          (P.einige n2)
      | ALLE n             (P.je_zwei n)
      | EINIGE n           (P.ein_paar n)

 iv : ivl                  (ivl)
      | tv refl            (fn t => (fn x => tv(t)(x,x)))

 stv : tv rezi             (fn t => (fn (x,y) =>
                                        B.und(tv(t)(x,y),tv(t)(y,x))))
      | SIND n             (fn t => (fn (x,y) => B.und(n(x),n(y))))

 advld : pl np             (fn t => pl(D.position np t))
```

Die Grammatik behandelt die Adverbiale recht grob: am Satzende können beliebig viele der (einstelligen) temporalen oder lokalen Adverbien stehen, wobei immer das letzte den weitesten Wirkungsbereich hat.

Außerdem hängt der Wahrheitswert nur bei Aussagen mit individuellem Subjekt und intransitivem Verb vom Ort der Äußerung ab. Da verschiedene Individuen sich nicht gleichzeitig am selben (punktförmigen) Ort befinden können, würde eine Ortsabhängigkeit bei (anders als durch $\exists$) quantifizierten Subjekten keine wahren Sätze ergeben.

Bemerkung 5.6 Der von ML-Yacc erzeugte Parser hat eine Reihe von (ca. 15) Shift/Reduce-Konflikten, bei denen standardmäßig (m.W.) eine Reduktion vorgenommen wird, d.h. man wendet eine Grammatikregel an, statt ein weiteres Eingabewort zu lesen. Manche Eingaben werden daher vielleicht anders analysiert als man erwartet, und folglich wird dann auch ein anderer Wert berechnet. Um eindeutige Analysen zu erzwingen, könnte man in der Grammatik Klammern um gewisse Teilausdrücke vorschreiben.

Das Format von ML-Yacc ist auch nicht geeignet, die Bedeutung eines Ausdrucks auf die Bedeutung eines transformierten Ausdrucks zurückzuführen.

5.4.4 Lexer mit Lokalpräpositionen und -Adverbien

Der Lexer für TNL muß um lokale Präpositionen und Ortsadverbien, die bisher nicht erlaubt waren, erweitert werden:

⟨*TNL/lnl.lex*⟩≡

```
structure P = D.P
structure Z = D.Z
structure O = D.O

structure Tokens = Tokens
structure Interface = Interface

open Interface

type pos = Interface.pos
type svalue = Tokens.svalue
type ('a,'b) token = ('a,'b) Tokens.token
type lexresult = (svalue,pos) token

fun eof() = Tokens.EOF(!line,!line)
fun makeInt (s : string) = s

%%
%header (functor NL_LexFun(structure Tokens: NL__TOKENS
                           structure Interface: INTERFACE) : LEXER);
%full

idchars=[0-9]|[_&+/@|#*‘’><];
alpha=[a-zA-ZÄÖÜäöüß];
comment=%(.*)\n;
whitespace=[\t\ ]*;
%%
<INITIAL>{whitespace} => (lex());
<INITIAL>"Der"    => (Tokens.DET(!line,!line));
```

```
<INITIAL>"Die"    => (Tokens.DET(!line,!line));
<INITIAL>"Jede"   => (Tokens.JEDER(!line,!line));
<INITIAL>"Jeder"  => (Tokens.JEDER(!line,!line));
{comment}  => (next_line(); lex());
\n         => (next_line(); lex());
"."        => (Tokens.PUNKT(!line,!line));
"und"      => (Tokens.UND(!line,!line));
"oder"     => (Tokens.ODER(!line,!line));
"nicht"    => (Tokens.NICHT(!line,!line));
"der"      => (Tokens.DET(!line,!line));
"die"      => (Tokens.DET(!line,!line));
"ist"      => (Tokens.IST(!line,!line));
"Anna"     => (Tokens.pn(P.Anna,!line,!line));
"Dieter"   => (Tokens.pn(P.Dieter,!line,!line));
"Emil"     => (Tokens.pn(P.Emil,!line,!line));
"Maria"    => (Tokens.pn(P.Maria,!line,!line));
"Mann"     => (Tokens.n(P.Mann,!line,!line));
"Frau"     => (Tokens.n(P.Frau,!line,!line));
"Student"  => (Tokens.n(P.Student,!line,!line));
"arbeitet" => (Tokens.ivl(P.arbeitet,!line,!line));
"kennt"    => (Tokens.tv(P.kennt,!line,!line));
"liebt"    => (Tokens.tv(P.liebt,!line,!line));
"jede"     => (Tokens.JEDER(!line,!line));
"jeder"    => (Tokens.JEDER(!line,!line));
"eine"     => (Tokens.EIN(!line,!line));
"ein"      => (Tokens.EIN(!line,!line));
"alle"     => (Tokens.ALLE(!line,!line));
"einige"   => (Tokens.EINIGE(!line,!line));
"sich"     => (Tokens.refl(!line,!line));
"einander" => (Tokens.rezi(!line,!line));
"Nachbarn" => (Tokens.n2(P.Nachbar,!line,!line));
"Brüder"   => (Tokens.n2(P.Bruder,!line,!line));
"Männer"   => (Tokens.n(P.Mann,!line,!line));
"Frauen"   => (Tokens.n(P.Frau,!line,!line));
"Studenten"  => (Tokens.n(P.Student,!line,!line));
"arbeiten" => (Tokens.ivl(P.arbeitet,!line,!line));
"kennen"   => (Tokens.tv(P.kennt,!line,!line));
"lieben"   => (Tokens.tv(P.liebt,!line,!line));
"sind"     => (Tokens.SIND(!line,!line));
"gestern"  => (Tokens.advti(Z.gestern,!line,!line));
"heute"    => (Tokens.advti(Z.heute,!line,!line));
"morgen"   => (Tokens.advti(Z.morgen,!line,!line));
"immer"    => (Tokens.advtq(Z.immer,!line,!line));
"nie"      => (Tokens.advtq(Z.nie,!line,!line));
"irgendwann" => (Tokens.advtq(Z.irgendwann,!line,!line));
"bis"      => (Tokens.advtq2(Z.bis,!line,!line));
```

```
"seit"     => (Tokens.advtq2(Z.seit,!line,!line));
"vor"      => (Tokens.pl(O.vor,!line,!line));
"hinter"   => (Tokens.pl(O.hinter,!line,!line));
"neben"    => (Tokens.pl(O.neben,!line,!line));
"irgendwo" => (Tokens.advlq(O.irgendwo,!line,!line));
"nirgendwo"=> (Tokens.advlq(O.nirgendwo,!line,!line));
"überall"  => (Tokens.advlq(O.ueberall,!line,!line));
"hier"     => (Tokens.advli(O.hier,!line,!line));
.          => (error ("ignoring illegal character " ^ yytext,
                      !line,!line); lex());
```

5.4.5 Automatische Erstellung des Programmpakets für LNL

Alle Dateien für `LNL` werden im Verzeichnis `TNL` abgelegt. Um das Paket benutzen zu können, muß man analog zur Erstellung von TNL vorgehen und dabei statt `tnl.*.cm` die Datei `lnl.*.cm` verwenden.

Die Beschreibungsdatei, die der Compilation Manager (mit SML/NJ version 110.42) dazu braucht, ist

⟨*TNL/lnl.110.42.cm*⟩≡

```
Library
  structure Zeit
  structure Ort
  structure Boole
  structure Personen
  structure NL
is
  $/smlnj-lib.cm
  $/ml-yacc-lib.cm
  $/basis.cm

  boole.sml
  quantoren.sml
  zeit.sml
  ort.sml
  personen.sml

  lnl.sml
  lnl.lex          (* fuer Tokenizer *)
  lnl.grm          (* fuer Parser *)

  nl.interface.sml (* fuer Lexer und Linker *)
  nl.link.sml      (* an LMU: nl.link.110.0.3.sml *)
```

An den Rechnern der LMU-Informatik muß man wieder mit `sml-110.0.42` arbeiten und in der letzten Zeile `nl.link.110.0.3.sml` benutzen.

5.4.6 Beispiele

Nach den Definitionen aus `Personen` und `Ort` befindet sich Maria heute (zur Zeit `t = 10`) an der Position `(5,10)` und gestern an der Position `(5,4)`, während sich Emil heute an der Position `(10,10)` und gestern an der Position `(9,9)` befindet. Da Emil zu jedem Zeitpunkt arbeitet, und `(10,10) neben (5,10)` gilt, erhält man:

⟨*Beispiel*⟩≡
```
- NL.parse();
Emil arbeitet neben Maria.
Value is: true
Emil arbeitet neben Maria gestern.
Value is: false
```

Analog erhält man:

⟨*Beispiel*⟩+≡
```
Maria arbeitet vor Anna.
Value is: false
Maria arbeitet vor Anna morgen.
Value is: false
Maria arbeitet vor Anna gestern.
Value is: true
```

⟨*Beispiel*⟩+≡
```
- NL.parse();
Emil arbeitet irgendwo.
Value is: true
Emil arbeitet.
Value is: false

Anna arbeitet.
Value is: true
Anna arbeitet überall.
Value is: false
```

Da die Wahrheitswerte von Aussagen vom Ort des Subjekts abhängen, erhalten wir manchmal seltsame Werte:

⟨*Beispiel*⟩+≡
```
Emil kennt sich irgendwo.
Value is: true
Emil kennt sich.
Value is: false
```

5.5 Indexikalische Ausdrücke

Indexikalische Ausdrücke vom Typ σ zur Bezeichnung

(a) von Sprecher und Hörer(n) wie *ich*, *du*, *wir*, *ihr*,

(b) von Orten wie *hier*, *dort*, *neben*, *hinter*,

(c) von Richtungen wie *hierher*, *nach rechts*, *von oben*, oder

(d) von Zeitpunkten wie *jetzt*, *morgen*, *vor drei Jahren*

werden in einer Interpretation $\mathcal{D}$ nicht durch feste Werte der entsprechenden Bereiche D_σ interpretiert, sondern als Werte in diesen Bereichen, die von einem *Kontext der Äußerung* abhängen. Die Bedeutungen der indexikalischen Wörter w sind dann Funktionen

$$w^{\mathcal{D}} : \textit{Äußerungskontext} \to D_\sigma.$$

Ein solcher Kontext kann durch Belegungungen $g_\sigma : \textit{Index}_\sigma \to D_\sigma$ der indexikalischen Wörter der Kategorie σ durch Werte aus D_σ dargestellt werden.

Der Wert eines beliebigen Ausdrucks α wird dann relativ zu einer solchen Belegung $g = \langle g_\sigma \mid \sigma \textit{ ein Typ} \rangle$ der indexikalischen Wörter und einer Belegung $h = \langle h_\sigma : \textit{Var}_\sigma \to D_\sigma \rangle$ der Variablen bestimmt:

$$[\![\alpha]\!]^{\mathcal{D}}_g(h).$$

Das haben wir oben für den Fall der indexikalischen Zeitadverbiale durchgeführt.[32]

Wie wir dort bemerkt hatten, ist die Äußerungssituation s, die wir durch die Belegung g der indexikalischen Wörter darstellen, auch dann nötig, wenn α keine indexikalischen Wörter enthält, z.B. weil der Wert einer Aussage von der Tempusform der Verben abhängig sein soll, und deren „Bedeutung" ist relativ zum Äußerungszeitpunkt.

Die Belegungen der indexikalischen Ausdrücke werden aber nicht benutzt, um Aussagen über die Menge der möglichen Werte zu machen, wie es bei den quantifizierenden Temporaladverbien oder -Operatoren der Fall war.

Manche indexikalischen Ausdrücke erhalten ihre Bedeutung durch die Ausführung einer Geste während des Sprechens, etwa das *dort* durch einen Fingerzeig[33].

Bem: Es gibt auch indexikalische Adverbien, z.B. *Leider ist das Wetter umgeschlagen.* = der Sprecher bedauert, daß das Wetter umgeschlagen ist.

[32] Bei den Ortsadverbien haben wir nur *hier* indexikalisch behandelt, während *neben* und *hinter* absolut behandelt wurden, da wir keine Sprecherposition berücksichtigt haben.

[33] Die Geste ist allerdings auch ein Zeichen, das man erst einmal richtig deuten muß!

Literatur

[1] Aristoteles. *Kategorien/Hermeneutik. Organon Band 2.* Felix Meiner Verlag, 1998. Griechisch/Deutsch.

[2] George H. Boole. *An Investigation of the Laws of Thought.* Macmillan, London, 1854.

[3] Michael Böttner. *Relationale Grammatik.* Max Niemeyer Verlag, Tübingnen, 1999.

[4] David R. Dowty, Robert E. Wall, and Stanley Peters. *Introduction to Montague Semantics.* D. Reidel, Dordrecht, 1981.

[5] Gottlob Frege. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, NF 100:25–50, 1892.

[6] Richard Montague. The proper treatment of quantification in ordinary english. In Richmond Thomason, editor, *Formal Philosophy. Selected Papers of Richard Montague.* Yale Univ. Press, 1974.

[7] David R. Tarditi and Andrew W. Appel. *ML-Yacc User's Manual. Version 2.3*, 1994.