Diskrete Strukturen

Helmut Schwichtenberg

Mathematisches Institut der Universität München
Wintersemester 2002/2003

Inhaltsverzeichnis

Vorwort iii

Kapitel 1. Aussagenlogik 1
1. Formeln der Aussagenlogik 2
2. Beweise 3
3. Intuitionistische und klassische Logik 6
4. Konjunktive und disjunktive Normalform 10

Kapitel 2. Quantorenlogik 13
1. Formale Sprachen 14
2. Beweise 16
3. Intuitionistische und klassische Logik 19
4. Konjunktive, disjunktive und pränexe Normalformen 21
5. Starke Disjunktion und Existenz 24
6. Rechnergestütztes Beweisen 26

Kapitel 3. Logik für freie Algebren 33
1. Typen und Terme 33
2. Formeln 42
3. Axiome 43
4. Beispiele für Induktionsbeweise 44

Kapitel 4. Relationen 51
1. Allgemeine Begriffe 51
2. Verknüpfung von Relationen und Matrizenmultiplikation 51
3. Transitive Hülle: der Warshall-Algorithmus 55
4. Ein dem Warshall-Algorithmus entsprechender Beweis 57

Kapitel 5. Graphen 61
1. Allgemeine Begriffe 61
2. Eulersche Wege und Zyklen 65
3. Abstände in bewerteten Graphen 70
4. Ein dem Dijkstra-Algorithmus entsprechender Beweis 75

Kapitel 6. Bäume 81
1. Allgemeine Begriffe 81
2. Aufspannende Bäume in Graphen: der Kruskal-Algorithmus 83

Kapitel 7. Induktive Definitionen 87
1. Fixpunkte 87
2. Approximation von Fixpunkten 88
3. Induktive Definitionen mit endlich vielen Vorgängern 90

4. Rekursion 91

Kapitel 8. Endliche Approximation unendlicher Objekte 93
1. Σ_1-Formeln und Berechenbarkeit 93
2. Informationssysteme 95
3. Abbildungen zwischen Informationssystemen 98
4. Kartesische Produkte 103
5. Partielle stetige Funktionale 107
6. Berechenbare Funktionale 108

Anhang A. MPC-Beweis zum Warshall-Algorithmus 111

Anhang. Literaturverzeichnis 121

Anhang. Index 123

Vorwort

Gegenstand dieser Vorlesung sind Relationen, Graphen und Bäume mit wesentlichen zugehörigen Algorithmen, sowie Grundbegiffe der mathematischen Logik einschließlich des λ-Kalküls. Was haben beide Gebiete miteinander zu tun? Wir wollen zeigen, daß und wie die Logik zum Entwurf korrekter Programme beitragen kann. Wir werden also „angewandte“ Logik betreiben, nicht mathematische Logik, in der unter anderem logische Systeme analysiert werden; letztere ist Gegenstand einer eigenen Vorlesung im Hauptstudium.

In der Logik unterscheidet man „moderne“ Axiomensysteme, in der beliebige Modelle untersucht werden (wie etwa in der Gruppentheorie), und „klassische“ Axiomensysteme, die mit der Absicht aufgestellt werden, ein konkretes Modell möglichst genau zu beschreiben (wie etwa die PEANO-Axiome der natürlichen Zahlen). In einer Vorlesung über mathematische Logik zeigt man:

- reine Axiomensysteme sind vollständig, aber
- klassische Axiomensysteme sind unvollständig.

Da wir uns hier mit klassischen Axiomensystemen befassen wollen, haben Vollständigkeitssätze in dieser Vorlesung keinen Platz. Trotzdem ist es gut zu wissen, daß die reine Logik vollständig ist.

Was hat die Logik mit Programmentwicklung zu tun? Ein Beweis von $\forall x \exists y A(x, y)$ (mit dem starken oder konstruktiven Existenzquantor $\exists$) „enthält“ ein Programm, das aus dem Beweis ablesbar ist. Im allgemeinen ist dies ein funktionales Programm. Gängige funktionale Programmsprachen sind LISP, der LISP-Dialekt SCHEME, ML, HASKELL und MIRANDA. Wir werden uns hier auf SCHEME beziehen. Ein naheliegender Einwand gegen diesen Ansatz, Programme aus Beweisen zu extrahieren, besteht in der Beobachtung, daß die Idee eines Algorithmus im allgemeinen auch am Anfang eines konstruktiven Beweises steht; warum implementiert man dann diesen Algorithmus nicht direkt? Hierzu läßt sich folgendes sagen.

- Es ist einfach mechanisch nachzuprüfen, ob ein gegebener Beweis korrekt ist. Ferner ist aus einem Beweis das Programm mechanisch extrahierbar. Man erhält also ein aufgrund seiner Konstruktion korrektes Programm.
- Wiederverwendung: Beweise sind (im Gegensatz zu vielen Programmen) verständlich; sie lassen sich deshalb relativ leicht an veränderte Situationen anpassen. Dasselbe gilt dann auch für die extrahierten Programme.
- Partielle Auswertung/Pruning: Es ist möglich, daß die als Beweise gegebenen und deshalb „extrem kommentierten“ Programme bei

Vorliegen von zusätzlichen Informationen über die Daten verändert werden können. Im Fall mehrerer Lösungen eines Programmierproblems kann sich dabei auch die erhaltene Lösung verändern.

In dieser Vorlesung soll ein Streifzug durch die genannten Gebiete versucht werden. Ich erhoffe mir, daß die Studenten am Ende

- wissen, was ein vollständiger, formaler Beweis ist, und etwas Erfahrung gewonnen haben, wie man formale Beweise erstellen kann.
- die wichtigsten Begriffe und Algorithmen über Relationen, Graphen und Bäume kennengelernt haben, und ferner
- zum Thema Programmextraktion aus Beweisen wichtige theoretische Konzepte kennengelernt und anhand einfacher Beispiele auch etwas Erfahrung gewonnen haben.

Da man am besten durch Tun lernt, wird im Rahmen der Übungen ausreichend Gelegenheit geboten, das Erstellen von Beweisen mit Rechnerunterstützung zu erlernen. Zu diesem Zweck wird ein eigens entwickelter Beweisprüfer MPC „Minlog Proof Checker“ verwendet, der eine Textdatei daraufhin überprüft, ob sie einen korrekten Beweis darstellt, und sie mit entsprechenden Kommentaren versieht.

Verwendet habe ich neben eigenen Vorarbeiten unter anderem die Bücher von THOMAS IHRINGER [**3**] (*Diskrete Mathematik*, Teubner, Stuttgart, 1994), KENNETH H. ROSEN [**7**] (*Discrete Mathematics and its Applications*, Random House, New York, 1988) und REINHARD DIESTEL [**1**] (*Graphentheorie*, Springer, Heidelberg, 2000), sowie Vorlesungen meiner Kollegen WILFRIED BUCHHOLZ und OTTO FORSTER (letztere in der Form einer von TANJA ZIMMERMANN angefertigten Ausarbeitung).

Bedanken möchte ich mich bei LAURA CROSILLA und MARTIN KÜBLER für ihre Hilfe bei der Vorbereitung der Übungen und dem Anfertigen der Musterlösungen. Einen besonderen Dank schulde ich MARTIN RUCKERT, der wesentliche Teile des in dieser Vorlesung verwendeten MPC-Systems konzipiert und implementiert hat.

München, Februar 2003
Helmut Schwichtenberg

KAPITEL 1

Aussagenlogik

Untersuchungsgegenstand der ersten Teile der Vorlesung sind mathematische Beweise. Zunächst befassen wir uns mit den Grundlagen der Formalisierung derartiger Beweise; in diesem einführenden Kapitel mit der Aussagenlogik.

Zur Darstellung von Beweisen verwenden wir ein System des „natürlichen Schließens“, das 1934 von GENTZEN [**2**] eingeführt wurde. Es gibt zwei Hauptgründe für diese Wahl. Erstens handelt es sich – wie schon der Name sagt – um ein *natürliches* Beweissystem, in dem Beweise in einer Art dargestellt sind, wie sie ein sorgfältiger Mathematiker formulieren würde, wenn er alle Einzelheiten einer Argumentation aufschreiben möchte. Zweitens sind formale Beweise im natürlichen Schließen eng verknüpft (mittels der sogenannten CURRY-HOWARD Korrespondenz) mit Termen im getypten λ-Kalkül. Dies liefert nicht nur eine kompakte Schreibweise für formale Beweise (die sonst leicht zu unhandlichen Baumstrukturen werden), sondern öffnet auch einen Weg zur Anwendung von im Kontext des λ-Kalküls gebräuchlichen rechnerischen Techniken.

Außer der klassischen Logik werden wir uns auch mit konstruktiven Logiken befassen: mit Minimallogik und mit intuitionistischer Logik. Dies wird einige interessante Aspekte von Beweisen zum Vorschein bringen. So ist es zum Beispiel möglich, eine nur mit der Implikation formulierte (insbesondere negationsfreie) Formel der Aussagenlogik anzugeben, deren Beweis klassische Logik erfordert.

Eine besondere Eigenart von GENTZENs Kalkül des natürlichen Schließens ist es, daß für jede logische Verknüpfung *Einführungs-* und *Beseitigungsregeln* vorhanden sind. Dies führt auf das System der *Minimallogik*, das von JOHANSSON [**4**] 1937 eingeführt wurde. Gibt man dann dem speziellen Aussagensymbol $\perp$ (gelesen „falsum“, für „Falschheit“) einen besonderen Status, so erhält man die intuitionistische Logik (mittels des Prinzips des ex-falso-quodlibet) und die klassische Logik (mittels des Prinzips des indirekten Beweisens)). Für diesen Aufbau der Logik ist es wichtig, daß wir uns auf eine Sprache beschränken, die nur die logischen Verknüpfungen $\rightarrow$ und $\wedge$ (und später $\forall$) enthält; die klassische Disjunktion (und später der klassische Existenzquantor) können mittels ihrer DE MORGANschen Definitionen eingeführt werden. Für diese Verknüpfungen lassen sich dann die üblichen Einführungs- und Beseitigungsregeln herleiten.

1. Formeln der Aussagenlogik

Eine *Aussage* ist nach ARISTOTELES ein *sprachliches Gebilde, von dem es sinnvoll ist zu sagen, es sei wahr oder falsch.* Beispiel: „Der Mond besteht aus grünem Käse“.

Wir betrachten eine formale Sprache, in der Aussagen durch *Aussagensymbole* wiedergegeben werden. Man nennt Aussagensymbole auch *atomare Formeln.* Aus ihnen kann man mittels *logischer Verknüpfungen* weitere Formeln zusammensetzen. Wir werden uns mit den folgenden logischen Verknüpfungen befassen.

$\wedge$	und	Konjunktion
$\rightarrow$	wenn – so	Implikation
$\neg$	nicht	Negation
$\vee$	oder (nicht-ausschließend)	Disjunktion
$\leftrightarrow$	genau dann, wenn	Äquivalenz

Allgemein ist mit A und B auch die Konjunktion $A \wedge B$ (gelesen „A und B“) eine Formel, und mit A und B auch die Implikation $A \rightarrow B$ (gelesen „wenn A, so B“) eine Formel. Die Äquivalenz $A \leftrightarrow B$ (gelesen „A genau dann wenn B“) ist definiert durch

$$A \leftrightarrow B := (A \rightarrow B) \wedge (B \rightarrow A).$$

Später (bei der Einführung der intuitionistischen Logik) werden wir ein spezielles Aussagensymbol $\perp$ (gelesen „falsum“) auszeichnen. Damit definieren wir die Negation $\neg A$ (gelesen „nicht A“) und die Disjunktion (gelesen „A oder B“) durch

$$\begin{aligned} \neg A \quad &:= A \rightarrow \perp, \\ A \vee B &:= \neg(\neg A \wedge \neg B). \end{aligned}$$

BEMERKUNG. Die Wahrheit oder Falschheit von $A \wedge B$ hängt allein von der Wahrheit oder Falschheit von A und von B ab, nicht von der Bedeutung dieser Aussagen. Genauso ist es mit den anderen logischen Verknüpfungen.

BEMERKUNG. Man beachte, daß „wenn – so“ im mathematischen Sinn gemeint ist, d.h. es braucht kein kausaler Zusammenhang zwischen A und B zu bestehen.

Klammern werden so gesetzt, daß sich die Struktur einer Formel eindeutig ergibt. Um Klammern zu sparen, vereinbaren wir folgendes.

(a) Der Wirkungsbereich von $\neg$ ist so klein wie möglich. Beispiel:

$$\neg A \wedge B \text{ meint } (\neg A) \wedge B, \text{ nicht } \neg(A \wedge B).$$

(b) Der Wirkungsbereich von $\wedge$ und $\vee$ ist so klein wie möglich, unter Berücksichtigung von (a). Beispiel:

$$\neg A \wedge B \rightarrow C \text{ meint } ((\neg A) \wedge B) \rightarrow C.$$

(c) Bei wiederholter Anwendung einer Verknüpfung ist stets Rechtsklammerung gemeint. Beispiel:

$$A \rightarrow B \rightarrow C \text{ meint } A \rightarrow (B \rightarrow C), \text{ nicht } (A \rightarrow B) \rightarrow C.$$

$$(A \wedge B \rightarrow C) \rightarrow A \rightarrow B \rightarrow C \text{ meint } (A \wedge B \rightarrow C) \rightarrow (A \rightarrow (B \rightarrow C)).$$

2. Beweise

Wir wollen jetzt den Begriff einer *logischen Herleitung* einer Formel A definieren. Dazu verwenden wir ein System des „natürlichen Schließens“, das 1934 von GENTZEN eingeführt wurde. Seine besondere Eigenart ist es, daß für jede logische Verknüpfung *Einführungs- und Beseitigungsregeln* vorhanden sind.

Wir beginnen mit einigen Beispielen für natürliche Beweise. Für die Klammerung beachte man die gerade eingeführten Klammerkonventionen, insbesondere obiges Beispiel (c).

$$(A \wedge B \to C) \to A \to B \to C. \tag{1}$$

BEWEIS. Gelte $A \wedge B \to C$. Zu zeigen: $A \to B \to C$. Gelte also A. Zu zeigen: $B \to C$. Gelte also B. Zu zeigen: C. Wir haben $A \wedge B$, nach den letzten beiden Annahmen. Also auch C, nach der ersten Annahme. □

$$(A \to B \to C) \to A \wedge B \to C. \tag{2}$$

BEWEIS. Gelte $A \to B \to C$. Zu zeigen: $A \wedge B \to C$. Gelte also $A \wedge B$. Zu zeigen: C. Wir haben A, nach der letzten Annahme. Also auch $B \to C$, nach der ersten Annahme. Wir haben B, wieder nach der letzten Annahme. Daher auch C, nach den letzten beiden Aussagen. □

Ein Charakteristikum dieser Beweise ist es, daß Annahmen eingeführt und wieder beseitigt werden: Zu jedem Zeitpunkt im Beweis kennt man die jetzt freien Annahmen.

Wir reservieren das Wort *Beweis* für einen informalen Kontext (oder die „Meta-Stufe“); eine formale Darstellung eines Beweises nennen wir *Herleitung* oder *Ableitung*.

Eine anschauliche Art, Herleitungen zu definieren, besteht darin, sie als beschriftete Bäume aufzufassen. Die Beschriftungen sind Formeln. An der Wurzel des Baums befindet sich die Konklusion der gesamten Herleitung. Im natürlichen Schließen arbeitet man mit *Annahmen*, die an Blättern des Baums stehen; sie können *offen* oder *geschlossen* (man sagt auch: *gestrichen*) sein.

Jede dieser Annahmen ist mit einer *Marke* versehen. Als Marken verwenden wir *Annahmenvariablen* $\Box_0, \Box_1, \ldots$; Mitteilungszeichen für Annahmenvariablen sind $u, v, w, u_0, u_1, \ldots$. Wenn an einer späteren Stelle (d.h. an einem Knoten im Baum unterhalb einer solchen Annahme) die Abhängigkeit von dieser Annahme beseitigt wird, notieren wir dies durch Angabe der Annahmenvariablen. Da wir dieselbe Annahme auch mehrfach verwenden können — dies war etwa im Beispiel (2) der Fall —, darf in einem Baum eine mit u markierte Annahme A (mitgeteilt durch $u\colon A$) auch mehrfach vorkommen. Wir verlangen jedoch, daß verschiedene Annahmeformeln stets verschiedene Marken bekommen.

Einen inneren Knoten im Baum verstanden wir als Resultat eines Übergangs von *Prämissen* zu einer *Konklusion*. Die zulässigen Übergänge werden

durch die *Regeln* bestimmt. Die Beschriftung des Knotens enthält dann neben der Konklusion noch den Namen der verwendeten Regel. In manchen Fällen bindet eine Regel eine Annahmenvariable u (und beseitigt damit die Abhängigkeit von allen darüber stehenden, mit u markierten Annahmen $u\colon A$). Dann wird die abgebundene Annahmenvariable der Beschriftung hinzugefügt.

Regeln des natürlichen Schließens. Zunächst haben wir eine Annahmeregel, die es gestattet, eine beliebige mit einer Marke u versehene Formel A als Annahme hinzuschreiben:

$$u\colon A \quad \text{Annahme}$$

Die restlichen Regeln gliedern sich in Einführungs- und Beseitigungsregeln für die logischen Verknüpfungen $\wedge$ und $\to$. Für die Konjuktion $\wedge$ haben wir eine Einführungsregel $\wedge^+$ und zwei Beseitigungsregeln $\wedge_0^-$ und $\wedge_1^-$.

$$\dfrac{\begin{array}{c}\mathcal{D}_0\\ A\end{array}\quad\begin{array}{c}\mathcal{D}_1\\ B\end{array}}{A\wedge B}\,\wedge^+ \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ A\wedge B\end{array}}{A}\,\wedge_0^- \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ A\wedge B\end{array}}{B}\,\wedge_1^-$$

Für die Implikation $\to$ gibt es eine Einführungsregel $\to^+u$ und eine Beseitigungsregel $\to^-$, die man auch *modus ponens* nennt. Die linke Prämisse $A\to B$ in $\to^-$ nennt man *Hauptprämisse*, die rechte Prämisse A *Nebenprämisse*. Man beachte, daß bei Anwendung einer $\to^+u$-Regel *alle* darüber stehenden mit u markierten Annahmen A gestrichen werden.

$$\dfrac{\begin{array}{c}[u\colon A]\\ \mathcal{D}\\ B\end{array}}{A\to B}\,\to^+u \qquad \dfrac{\begin{array}{c}\mathcal{D}_0\\ A\to B\end{array}\quad\begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B}\,\to^-$$

Wir geben Herleitungen für die oben bewiesenen Formeln (1) – (2) an. Da meist die verwendete Regel durch die an den Knoten stehenden Formeln bestimmt ist, verzichten wir im allgemeinen auf die Angabe der Regel.

$$\dfrac{\dfrac{\dfrac{\dfrac{u\colon A\wedge B\to C \qquad \dfrac{v\colon A \qquad w\colon B}{A\wedge B}}{C}}{B\to C}\,\to^+w}{A\to(B\to C)}\,\to^+v}{(A\wedge B\to C)\to(A\to(B\to C))}\,\to^+u \tag{1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{u\colon A\to(B\to C) \qquad \dfrac{v\colon A\wedge B}{A}}{B\to C} \qquad \dfrac{v\colon A\wedge B}{B}}{C}}{A\wedge B\to C}\,\to^+v}{(A\to(B\to C))\to(A\wedge B\to C)}\,\to^+u \tag{2}$$

Wir schreiben $\vdash A$ und nennen A *herleitbar* (in der *Minimallogik*), wenn es eine Herleitung von A ohne freie Annahmen gibt. Eine Formel B heißt herleitbar aus den Annahmen $A_1,\ldots,A_n$, wenn es eine Herleitung mit freien Annahmen unter $A_1,\ldots,A_n$ gibt. Sei Γ eine (endliche oder unendliche)

Menge von Formeln. Wir schreiben $\Gamma \vdash B$, wenn die Formel B aus endlich vielen Annahmen $A_1, \ldots, A_n \in \Gamma$ herleitbar ist.

Lineare Formulierung des natürlichen Schließens. Man kann eine im Kalkül des natürlichen Schließens vorliegende Herleitung offenbar auch in linearisierter Form mitteilen. In jeder Zeile steht dann eine Formel, und man muß dazuschreiben, ob es sich um eine Annahme handelt, oder ob diese Formel mit Hilfe einer Regel aus vorangehenden Formeln erschlossen wurde. Um die Möglichkeit Einführens und des Streichens von Annahmen wiederzugeben, verwenden wir eine *Blockstruktur*.

Ein *Block* beginnt mit { und endet mit }. Nach der öffnenden geschweiften Klammer folgt eine Formel. Damit wird eine für den Block lokale Annahme festgelegt. Nach dem Schließen des Blocks ist dann eine Implikation bewiesen. Beispiel:

```
MPC;
PRED . A B C;
PROOF;
{ A & B -> C.
  { A.
    { B.
      A & B;
      C;
    }
    B -> C;
  }
  A -> B -> C;
}
(A & B -> C) -> A -> B -> C;
```

Man kann alles dies in eine Textdatei schreiben und vom Rechner überprüfen lassen; für diesen Zweck ist ein Beweisprüfer MPC („Minlog Proof Checker“) bereitgestellt. Er überprüft die Korrektheit der Angaben und kommentiert sie in angemessener Weise. Im Beispiel erhält man:

```
PRED . A B C;
; ok, predicate constant A: (arity) added
; ok, predicate constant B: (arity) added
; ok, predicate constant C: (arity) added
PROOF;; initializing mpc
{ A & B -> C.   // 0 assumed.
  { A.  // 1 assumed.
    { B.        // 2 assumed.
      A & B;    // ok, 3 proved by and-intro from 1 and 2
      C;        // ok, 4 proved by imp-elim from 0 and 3
    }   // ok, 5 B -> C proved.
    B -> C;     // ok, 6 proved by 5
  }     // ok, 7 A -> B -> C proved.
  A -> B -> C;  // ok, 8 proved by 7
}       // ok, 9 (A & B -> C) -> A -> B -> C proved.
(A & B -> C) -> A -> B -> C;    // ok, 10 proved by 9
```

3. Intuitionistische und klassische Logik

In der Minimallogik spielen alle Aussagensymbole dieselbe Rolle. Wir wollen jetzt ein spezielles Aussagensymbol auszeichnen: $\bot$, gelesen „falsum“. In unserer $\to\wedge$-Sprache erhalten wir dann die *intuitionistische Logik*, indem wir gewisse zusätzliche Annahmen verwenden, und zwar die sogenannten *Ex-Falso-Quodlibet*-Formeln (oder „Axiome“) Efq_P für jedes von $\bot$ verschiedene Aussagensymbol P

$$\bot \to P. \tag{Efq_P}$$

Ähnlich erhält man die *klassische Logik*: wir nehmen für jedes von $\bot$ verschiedene Aussagensymbol P das *Prinzip des indirekten Beweisens* für P als zusätzliche Annahme hinzu, also die Formel

$$\neg\neg P \to P; \tag{Stab_P}$$

diese Formel bezeichnet man auch als *Stabilität* von P.

Man beachte, daß mit $\bot$ für P beide Formeln trivialerweise herleitbar sind; z.B. für die Stabilität haben wir $\neg\neg\bot \to \bot = ((\bot \to \bot) \to \bot) \to \bot$. Die gesuchte Herleitung ist

$$\dfrac{v\colon (\bot \to \bot) \to \bot \qquad \dfrac{u\colon \bot}{\bot \to \bot}\to^+ u}{\bot}$$

Sei

$$\begin{aligned}\mathsf{Efq} &:= \{\, \mathsf{Efq}_P \mid P \text{ Aussagensymbol} \neq \bot \,\},\\ \mathsf{Stab} &:= \{\, \mathsf{Stab}_P \mid P \text{ Aussagensymbol} \neq \bot \,\}.\end{aligned}$$

Wir nennen die Formel A *klassisch (intuitionistisch) herleitbar* und schreiben $\vdash_c A$ ($\vdash_i A$), wenn es eine Herleitung von A aus Stabilitätsannahmen Stab_P (Ex-Falso-Quodlibet Annahmen Efq_P) gibt. Ebenso definieren wir klassische (intuitionistische) Herleitbarkeit aus Γ und schreiben $\Gamma \vdash_c A$ ($\Gamma \vdash_i A$), also

$$\begin{aligned}\Gamma \vdash_i A \quad &:\Longleftrightarrow \quad \Gamma \cup \mathsf{Efq} \vdash A,\\ \Gamma \vdash_c A \quad &:\Longleftrightarrow \quad \Gamma \cup \mathsf{Stab} \vdash A.\end{aligned}$$

Lemma 3.1 (Ex-falso-quodlibet). *Für jede Formel A gilt $\vdash_i \bot \to A$.*

Beweis. Durch Induktion über A konstruieren wir für jede Formel A eine Herleitung $\mathcal{D}_A$ von $\bot \to A$. *Fall* A Aussagensymbol. Mit Efq_A. *Fall* $A \wedge B$.

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\mathcal{D}_A\\ \bot \to A\end{matrix} \qquad u\colon \bot}{A} \qquad \dfrac{\begin{matrix}\mathcal{D}_B\\ \bot \to B\end{matrix} \qquad u\colon \bot}{B}}{A \wedge B}}{\bot \to A \wedge B}\to^+ u$$

Fall $A \to B$.

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\mathcal{D}_B\\ \bot \to B\end{matrix} \qquad u\colon \bot}{B}}{A \to B}}{\bot \to A \to B}\to^+ u$$

Damit ist der Beweis abgeschlossen. $\square$

LEMMA 3.2 (Stabilität). *Für jede Formel A (unserer $\to\wedge$-Sprache) gilt* $\vdash_c \neg\neg A \to A$.

BEWEIS. Induktion über A. In den konstruierten Herleitungen lassen wir der Kürze halber Anwendungen von $(\to^+)$ am Schluß fort. *Fall* A Aussagensymbol. Mit Stab_A.

Fall $A \wedge B$. Mit $\vdash (\neg\neg A \to A) \to (\neg\neg B \to B) \to \neg\neg(A \wedge B) \to A \wedge B$, was leicht aus $\vdash \neg\neg(A \wedge B) \leftrightarrow (\neg\neg A \wedge \neg\neg B)$ (Herleitung als Übung) folgt.

Fall $A \to B$. Mit $\vdash (\neg\neg B \to B) \to \neg\neg(A \to B) \to A \to B$. Eine Herleitung ist

$$\dfrac{u\colon \neg\neg B \to B \qquad \dfrac{v\colon \neg\neg(A \to B) \qquad \dfrac{\dfrac{u_1\colon \neg B \qquad \dfrac{\dfrac{u_2\colon A \to B \qquad w\colon A}{B}}{\bot}}{\neg(A \to B)} \to^+ u_2}{\bot}}{\neg\neg B} \to^+ u_1}{B}$$

Damit ist der Beweis abgeschlossen. □

LEMMA 3.3. $\Gamma \vdash A \Rightarrow \Gamma \vdash_i A$ *und* $\Gamma \vdash_i A \Rightarrow \Gamma \vdash_c A$.

BEWEIS. Es genügt zu zeigen, daß $\vdash_c \mathsf{Efq}_A$. Dies sieht man wie folgt.

$$\dfrac{\dfrac{\neg\neg A \to A \qquad \dfrac{u\colon \bot}{\neg\neg A} \to^+ v^{\neg A}}{A}}{\bot \to A} \to^+ u$$

Damit ist der Beweis abgeschlossen. □

Die Umkehrungen gelten jedoch nicht: Gegenbeispiele sind $\nvdash \bot \to A$ bzw. $\nvdash_i ((A \to B) \to A) \to A$ mit Aussagensymbolen A, B. (Ein Beweis dafür erfordert ein genaueres Studium der Herleitbarkeit.) Andererseits gilt offenbar $\vdash_i \bot \to A$, und man kann auch leicht eine klassische Herleitung der PEIRCE-Formel $((A \to B) \to A) \to A$ angeben (Übung).

LEMMA 3.4 (Fallunterscheidung). (a) $\vdash (A \to B) \to (\neg A \to B) \to \neg\neg B$. (b) $\vdash_c (A \to B) \to (\neg A \to B) \to B$.

BEWEIS. (a).

$$\dfrac{\dfrac{w\colon \neg B \qquad \dfrac{u_2\colon \neg A \to B \qquad \dfrac{\dfrac{w\colon \neg B \qquad \dfrac{u_1\colon A \to B \qquad v\colon A}{B}}{\bot}}{\neg A} \to^+ v}{B}}{\bot}}{\neg\neg B} \to^+ w$$

(b). Man verwende das Stabilitätslemma 3.2. □

Wir nennen zwei Formeln A und B *äquivalent* in der Minimallogik bzw. in der klassischen oder intuitionistischen Logik, wenn $\vdash A \leftrightarrow B$ bzw. $\vdash_c A \leftrightarrow B$ oder $\vdash_i A \leftrightarrow B$.

LEMMA 3.5 (Äquivalenzlemma). *Für* $\vdash_{mic} \in \{\vdash, \vdash_i, \vdash_c\}$ *gilt folgendes. Ist* $\vdash_{mic} A_1 \leftrightarrow A_2$ *und entsteht* B_2 *aus* B_1 *durch Ersetzen eines Teils* A_1 *von* B_1 *durch* A_2*, so gilt auch* $\vdash_{mic} B_1 \leftrightarrow B_2$.

BEWEIS. Induktion über B_1. Falls ganz B_1 ersetzt wird, so ist die Behauptung klar. Andernfalls muß B_1 eine zusammengesetzte Formel sein.

Fall $C_1 \wedge D_1$. Die Ersetzung finde etwa in C_1 statt. Zu zeigen ist dann $\vdash_{mic} C_1 \wedge D_1 \leftrightarrow C_2 \wedge D_1$. $\rightarrow$:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ C_1 \to C_2\end{array} \qquad \dfrac{C_1 \wedge D_1}{C_1}}{C_2} \qquad \dfrac{C_1 \wedge D_1}{D_1}}{C_2 \wedge D_1}$$

wobei $\mathcal{D}$ nach IH bekannt ist. $\leftarrow$ beweist man ähnlich.

Fall $C_1 \to D_1$. Falls die Ersetzung in C_1 stattfindet, ist zu zeigen $\vdash_{mic} (C_1 \to D_1) \leftrightarrow (C_2 \to D_1)$. $\rightarrow$:

$$\dfrac{\dfrac{C_1 \to D_1 \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ C_2 \to C_1\end{array} \qquad u\colon C_2}{C_1}}{D_1}}{C_2 \to D_1} \to^+ u$$

wobei wieder $\mathcal{D}$ nach IH bekannt ist. $\leftarrow$ beweist man ähnlich. Falls die Ersetzung in D_1 stattfindet, ist zu zeigen $\vdash_{mic} (C_1 \to D_1) \leftrightarrow (C_1 \to D_2)$. $\rightarrow$:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ D_1 \to D_2\end{array} \qquad \dfrac{C_1 \to D_1 \qquad u\colon C_1}{D_1}}{D_2}}{C_1 \to D_2} \to^+ u$$

wobei wieder $\mathcal{D}$ nach IH bekannt ist. $\leftarrow$ beweist man ähnlich. □

Einbettung der intuitionistischen und klassischen Logik in die Minimallogik. Nachdem wir die klassische und die intuitionistische Logik definiert haben, wollen wir jetzt zeigen, daß beide Logiken in die Minimallogik eingebettet werden können. Dies mag verwunderlich erscheinen; es folgt wesentlich aus der Tatsache, daß wir uns auf eine Sprache beschränkt haben, die nur die Verknüpfungen $\rightarrow, \wedge$ enthält.

Eine Formel A (unserer $\rightarrow\wedge$-Sprache) heißt *negativ*, wenn jedes Aussagensymbol $B \neq \bot$ in A negiert vorkommt, d.h. in einem Kontext $B \to \bot$.

LEMMA 3.6. *Für negative* A *gilt* $\vdash \neg\neg A \to A$.

BEWEIS. Dies beweist man wie das Stabilitätslemma 3.2 durch Induktion über A, wobei man $\vdash \neg\neg\neg B \to \neg B$ (Herleitung als Übung) anstelle der Stabilitätsannahmen verwendet. □

DEFINITION 3.7. (Negative Übersetzung ^{g} nach GÖDEL-GENTZEN).

$$\begin{aligned}
P^g &:= \neg\neg P \quad \text{für } P \text{ Aussagensymbol} \neq \bot,\\
\bot^g &:= \bot,\\
(A \wedge B)^g &:= A^g \wedge B^g,\\
(A \to B)^g &:= A^g \to B^g,
\end{aligned}$$

SATZ 3.8. *Für alle Formeln A gilt*

(a) $\vdash_c A \leftrightarrow A^g$,

(b) $\Gamma \vdash_c A$ *genau dann, wenn* $\Gamma^g \vdash A^g$, *wobei* $\Gamma^g := \{\, B^g \mid B \in \Gamma \,\}$.

BEWEIS. (a) folgt sofort aus dem Äquivalenzlemma 3.5. Für (b) ist die Richtung von rechts nach links klar. Für die andere Richtung argumentieren wir durch Induktion nach der klassischen Herleitung. Für eine Stabilitätsannahme $\neg\neg P \to P$ gilt $(\neg\neg P \to P)^g = \neg\neg\neg\neg P \to \neg\neg P$, und dies ist leicht herleitbar. *Fall* $\to^+$. Gelte

$$\dfrac{\begin{array}{c}[u\colon A]\\ \mathcal{D}\\ B\end{array}}{A \to B}\to^+ u$$

Dann haben wir nach IH

$$\begin{array}{c} u\colon A^g\\ \mathcal{D}^g\\ B^g\end{array} \quad \text{also} \quad \dfrac{\begin{array}{c}[u\colon A^g]\\ \mathcal{D}^g\\ B^g\end{array}}{A^g \to B^g}\to^+ u$$

Fall $\to^-$. Gelte

$$\dfrac{\begin{array}{c}\mathcal{D}_0\\ A \to B\end{array} \qquad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B}$$

Dann haben wir nach IH

$$\begin{array}{c}\mathcal{D}_0^g\\ A^g \to B^g\end{array} \quad \text{und} \quad \begin{array}{c}\mathcal{D}_1^g\\ A^g\end{array}, \text{ also } \dfrac{\begin{array}{c}\mathcal{D}_0^g\\ A^g \to B^g\end{array} \qquad \begin{array}{c}\mathcal{D}_1^g\\ A^g\end{array}}{B^g}$$

Die restlichen Fälle behandelt man ähnlich. □

KOROLLAR 3.9 (Einbettung der klassischen Logik in die Minimallogik). *Für negative A gilt $\vdash_c A$ genau dann, wenn $\vdash A$.*

BEWEIS. Nach dem Satz haben wir $\vdash_c A$ genau dann, wenn $\vdash A^g$. Da A negativ ist, muß jedes Aussagensymbol $P \neq \bot$ in A negiert vorkommen, ist also in A^g dreifach negiert (als $\neg\neg\neg P$). Die Behauptung folgt mit dem Äquivalenzlemma 3.5 aus $\vdash \neg\neg\neg P \leftrightarrow \neg P$. □

Da jede Formel klassisch äquivalent zu einer negativen Formel ist, haben wir damit eine Einbettung der klassischen in die Minimallogik erreicht.

Man beachte, daß $\nvdash \neg\neg P \to P$ für P ein Aussagensymbol $\neq \bot$. Das Korollar gilt also nicht für alle Formeln.

Abgeleitete Regeln für die klassische Disjunktion. Die klassische Disjunktion hatten wir definiert durch

$$A \vee B := \neg(\neg A \wedge \neg B).$$

Herleitungen von Formeln mit $\vee$ lassen sich oft leichter finden, wenn man nicht auf diese Definition zurückgeht, sondern direkt mit „*abgeleiteten Regeln*“ für $\vee$ arbeitet. Darunter verstehen wir die folgenden Regeln. Es gibt

es zwei Einführungsregeln $\vee_0^+, \vee_1^+$ und eine Beseitigungsregel $\vee^-$.

$$\dfrac{\begin{array}{c}\mathcal{D}\\ A\end{array}}{A \vee B}\,\vee_1^+ \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ B\end{array}}{A \vee B}\,\vee_0^+ \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ A \vee B\end{array} \quad \begin{array}{c}[u\colon A]\\ \mathcal{D}_0\\ C\end{array} \quad \begin{array}{c}[v\colon B]\\ \mathcal{D}_1\\ C\end{array}}{C}\,\vee^- uv$$

Der doppelte Strich deutet hierbei an, daß es sich nicht um eine primitive Regel unserer Logik handelt, sondern daß man an dieser Stelle einen Herleitungsteil so einsetzen kann, daß eine korrekt gebildete Herleitung entsteht. Wir geben im folgenden für jede der abgeleiteten Regeln den einzusetzenden Herleitungsteil an. Man beachte, daß es sich um Herleitungen in der klassischen Logik handelt, da Stabilitätsannahmen verwendet werden.

$\vee_0^+$ ($\vee_1^+$ wird entsprechend behandelt):

$$\dfrac{\dfrac{\dfrac{u\colon \neg A \wedge \neg B \quad \begin{array}{c}\mathcal{D}\\ A\end{array}}{\neg A}}{\bot}}{\neg(\neg A \wedge \neg B)}\,\to^+ u$$

$\vee^-$:

$$\dfrac{\begin{array}{c}\mathcal{D}_{\mathsf{Stab}}\\ \neg\neg C \to C\end{array} \quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \neg(\neg A \wedge \neg B)\end{array} \quad \dfrac{\dfrac{w\colon \neg C \quad \begin{array}{c}[u\colon A]\\ \mathcal{D}_0\\ C\end{array}}{\bot}}{\neg A}\,\to^+ u \quad \dfrac{\dfrac{w\colon \neg C \quad \begin{array}{c}[v\colon B]\\ \mathcal{D}_1\\ C\end{array}}{\bot}}{\neg B}\,\to^+ v}{\neg A \wedge \neg B}}{\bot}}{\neg\neg C}\,\to^+ w}{C}$$

Als Alternative kann man ebenfalls den Komfort der abgeleiteten Regeln für $\vee$ erhalten, indem man die folgenden (ableitbaren) Schemata als Axiome benutzt.

$$A \to A \vee B, \quad B \to A \vee B.$$
$$(A \to C) \to (B \to C) \to A \vee B \to C.$$

4. Konjunktive und disjunktive Normalform

Wir wollen zeigen, daß sich jede Formel klassisch äquivalent so umformen läßt, daß sie zu einer Konjunktion von Disjunktionen (bzw. zu einer einer Disjunktion von Konjunktionen) aussagenlogisch unzerlegbarer Formeln wird. Wir führen deshalb zunächst iterierte Konjunktionen und Disjunktion ein durch

$$\bigwedge_{i=1}^{n} A_i := A_1 \wedge \cdots \wedge A_n,$$
$$\bigvee_{i=1}^{n} A_i := A_1 \vee \ldots \vee A_n.$$

Lemma 4.1 ($\wedge, \vee$ sind assoziativ, kommutativ und distributiv).

(a) $\vdash_c A \wedge (B \wedge C) \leftrightarrow (A \wedge B) \wedge C$.
(b) $\vdash_c A \vee (B \vee C) \leftrightarrow (A \vee B) \vee C$.

(c) $\vdash_c A \wedge B \leftrightarrow B \wedge A$.
(d) $\vdash_c A \vee B \leftrightarrow B \vee A$.
(e) $\vdash_c A \wedge (B \vee C) \leftrightarrow (A \wedge B) \vee (A \wedge C)$.
(f) $\vdash_c A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C)$.

Beweis. Übung. □

Lemma 4.2 (de Morgan-Äquivalenzen). (a)

$$\vdash_c \neg(A \wedge B) \leftrightarrow \neg A \vee \neg B,$$
$$\vdash_c \neg(A \vee B) \leftrightarrow \neg A \wedge \neg B.$$

(b) *($\rightarrow$ aus $\neg, \wedge$ bzw. $\neg, \vee$).*

$$\vdash_c (A \rightarrow B) \leftrightarrow \neg(A \wedge \neg B),$$
$$\vdash_c (A \rightarrow B) \leftrightarrow \neg A \vee B.$$

(c) *(Tertium non datur).* $\vdash_c A \vee \neg A$.

Beweis. (a). Die Behauptung ergibt sich aus der Definition von $\vee$, dem Stabilitätslemma 3.2 und dem Äquivalenzlemma 3.5.

(b). $\vdash_c (A \rightarrow B) \leftrightarrow \neg(A \wedge \neg B)$. $\rightarrow$.

$$\dfrac{\dfrac{v\colon A \wedge \neg B}{\neg B} \quad \dfrac{u\colon A \rightarrow B \quad \dfrac{v\colon A \wedge \neg B}{A}}{B}}{\bot}$$

$\leftarrow$.

$$\dfrac{\mathcal{D}_{\mathsf{Stab}} \quad \neg\neg B \rightarrow B \quad \dfrac{\dfrac{u\colon \neg(A \wedge \neg B) \quad \dfrac{v\colon A \quad w\colon \neg B}{A \wedge \neg B}}{\bot}}{\neg\neg B} \rightarrow^+ w}{\dfrac{B}{A \rightarrow B} \rightarrow^+ v}$$

$\vdash_c (A \rightarrow B) \leftrightarrow \neg A \vee B$. $\rightarrow$.

$$\dfrac{v\colon \neg\neg A \wedge \neg B \quad \dfrac{\dfrac{v\colon \neg\neg A \wedge \neg B}{\neg B} \quad \dfrac{u\colon A \rightarrow B \quad w\colon A}{B}}{\bot}}{\neg\neg A}$$
$$\dfrac{\neg\neg A \quad \dfrac{\bot}{\neg A} \rightarrow^+ w}{\bot}$$

$\leftarrow$. Hier kann man die abgeleitete Regel $\vee^-$ verwenden; ferner benötigt man das Ex-Falso-Quodlibet-Lemma 3.1.

(c). Dies ergibt sich sofort aus der Definition von $\vee$:

$$\dfrac{\dfrac{u\colon \neg A \wedge \neg\neg A}{\neg\neg A} \quad \dfrac{u\colon \neg A \wedge \neg\neg A}{\neg A}}{\bot}$$

Damit ist der Beweis abgeschlossen. □

Satz 4.3 (Konjunktive und disjunktive Normalform). *Für jede Formel A gilt folgendes.*

(a) *A ist klassisch äquivalent zu einer Formel der Gestalt*

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} A_{ij} \qquad \textit{(konjunktive Normalform),}$$

wobei die A_{ij} Aussagensymbole oder Negationen davon sind.

(b) *A ist klassisch äquivalent zu einer Formel der Gestalt*

$$\bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} A_{ij} \qquad \textit{(disjunktive Normalform),}$$

wobei wieder die A_{ij} Aussagensymbole oder Negationen davon sind.

Beweis. Wir zeigen (a) und (b) gemeinsam durch Induktion über die gegebene Formel, unter Verwendung des Äquivalenzlemmas 3.5.

Fall A Aussagensymbol. In diesem Fall ist nichts zu zeigen.

Fall $A \wedge B$. Für (a) verwendet man die konjunktiven Normalformen von A und von B. (b) ergibt sich aus den disjunktiven Normalformen von A und von B durch Ausdistribuieren.

Fall $A \to B$. Die Behauptung ergibt sich aus $\vdash_c (A \to B) \leftrightarrow \neg A \vee B$ und den de Morgan-Äquivalenzen, und zwar für (a) unter Verwendung der disjunktiven Normalform von A und der konjunktiven Normalform von B und für (b) unter Verwendung der konjunktiven Normalform von A und der disjunktiven Normalform von B, wobei in (a) noch auszudistribuieren ist. □

Eine konjunktive Normalform von A (die offenbar nicht eindeutig bestimmt ist) nennt man auch eine *Klauselform* von A. Da eine Konjunktion genau dann herleitbar ist, wenn dies für jedes Konjunktionsglied gilt, kann man sich also bei der Suche nach Herleitungen von Formeln auf sogenannte Klauseln, also Disjunktionen von Aussagensymbolen oder Negationen davon beschränken.

Man beachte, daß die Herstellung etwa der konjunktiven Normalform einer gegebenen Formel im allgemeinen exponentiell viele (in der Länge der Formel) Rechenschritte erfordert, da bei den Distributivgesetzen Formeln dupliziert werden.

KAPITEL 2

Quantorenlogik

Zur Formulierung mathematischer Sachverhalte ist die Sprache der Aussagenlogik offensichtlich ungeeignet: man braucht noch Quantoren „für alle“ und „es gibt“. Wir formulieren ein System des natürlichen Schließens für die minimale Quantorenlogik, und übertragen die bisherigen Überlegungen auf diese allgemeinere Situation.

Um auch allgemeine Aussagen mit Quantoren machen zu können, benötigen wir *Variablen* $x, y, z, \ldots$. Dabei stellen wir uns vor, daß die Variablen über die Elemente eines fest gegebenen Individuenbereichs laufen.

Mit A ist auch die Allformel $\forall x A$ (gelesen „für alle x gilt A“) eine Formel. Hierbei wird im allgemeinen A eine Formel sein, in der die Variable x vorkommt.

Ferner ist mit A auch die Existenzformel $\exists^{\mathsf{cl}} x A$ (gelesen „es gibt ein x mit A“) eine Formel. Man beachte, daß bei der üblichen *klassischen* (oder schwachen) Auffassung des „es gibt“ $\exists^{\mathsf{cl}} x A$ dasselbe bedeutet wie $\neg\forall x \neg A$, also $\exists$ auch durch $\forall$ und $\neg$ definiert werden kann. Wir werden auch noch eine konstruktive (oder starke) Auffassung des Existenzquantors besprechen (geschrieben $\exists x A$), bei der dies nicht mehr der Fall ist.

Man beachte, daß die Verwendung der Quantoren $\forall x$ und $\exists^{\mathsf{cl}} x$ die Unterscheidung zwischen *freien* und *gebundenen* Vorkommen von Variablen notwendig macht. So ist etwa in $\forall x(x \leq x + y)$ das zweite und dritte Vorkommen der Variablen x gebunden, aber das einzige Vorkommen der Variablen y frei. Die letzte Formel kann man unter Erhalt ihrer Bedeutung auch umschreiben in $\forall z(z \leq z + y)$, denn dies ist nur eine Umbenennung der gebundenen Variablen. Ein ähnliches Phänomen ist aus der Analysis bekannt, wo ebenfalls

$$\int_0^1 \sin(xy)\,dx = \int_0^1 \sin(ty)\,dt$$

ist; x bzw. t kommen hier gebunden und y frei vor.

Beispiele zur Formalisierung mathematischer Aussagen. Als erstes betrachten wir die Axiome der Gruppentheorie. Grundbegriffe sind die Gruppenverknüpfung $\circ$ (ein zweistelliges Funktionssymbol), die Einheit e (eine Konstante), die Inversenbildung $^{-1}$ (ein einstelliges Funktionssymbol) und schließlich die Gleichheit $=$ (ein zweistelliges Relationssymbol). Als Axiome haben wir

$$\begin{aligned}
&\forall x \forall y \forall z[x \circ (y \circ z) = (x \circ y) \circ z] && \text{(Assoziativität)},\\
&\forall x[e \circ x = x] && \text{(Linkseinheit)},\\
&\forall x[x^{-1} \circ x = e] && \text{(Linksinverses)}.
\end{aligned}$$

Zwei Mengen sind gleich, wenn sie dieselben Elemente enthalten:

$\forall$	für alle Mengen
$\exists^{\mathsf{cl}}$	es gibt eine Menge
$x \in y$	x ist ein Element von y
$x = y$	x ist gleich y

$$\forall x \forall y (\forall z. z \in x \leftrightarrow z \in y) \to x = y.$$

f ist stetig im Punkt a:

f	einstelliges Funktionssymbol
a, 0	reelle Zahlen
$\forall$	für alle reelle Zahlen
$\exists^{\mathsf{cl}}$	es gibt eine reelle Zahl
$x > y$ $(x < y)$	x ist größer (kleiner) als y
$d(x, y)$	der Abstand von x und y

$$\forall \varepsilon. \varepsilon > 0 \to \exists^{\mathsf{cl}} \delta. \delta > 0 \wedge \forall x. d(x, a) < \delta \to d(f(x), f(a)) < \varepsilon;$$

hierfür schreibt man zur Abkürzung meist

$$\forall \varepsilon > 0 \exists^{\mathsf{cl}} \delta > 0 \forall x. d(x, a) < \delta \to d(f(x), f(a)) < \varepsilon.$$

1. Formale Sprachen

Wir geben jetzt eine genaue Beschreibung der Sprachen der Logik erster Stufe.

Gegeben sei eine abzählbar unendliche Menge $\{ v_i \mid i \in \mathbb{N} \}$ von *Variablen*; Mitteilungszeichen für Variable sind x, y, z. Eine *Sprache* $\mathcal{L}$ *erster Stufe* ist bestimmt durch ihre *Signatur*. Darunter versteht man folgendes.

- Für jede natürliche Zahl $n \geq 0$ eine (eventuell leere) Menge $\mathsf{Rel}_{\mathcal{L}}^{(n)}$ von n-stelligen *Relationssymbolen* (oder auch *Prädikatensymbolen*). 0-stellige Relationssymbole heißen *Aussagensymbole*. Wir nehmen wieder an, daß ein spezielles Aussagensymbol $\perp$ (gelesen „falsum“) vorhanden ist.
- Für jede natürliche Zahl $n \geq 0$ eine (eventuell leere) Menge $\mathsf{Fun}_{\mathcal{L}}^{(n)}$ von n-stelligen *Funktionssymbolen*. Die 0-stelligen Funktionssymbole heißen *Konstante*.

Die Mengen $\mathsf{Rel}_{\mathcal{L}}^{(n)}$ und $\mathsf{Fun}_{\mathcal{L}}^{(n)}$ und die Menge der Variablen wollen wir als paarweise disjunkt voraussetzen.

Zum Beispiel ist die Sprache $\mathcal{L}_G$ der Gruppentheorie bestimmt durch die Signatur bestehend aus den Funktionssymbolen $\circ, e, ^{-1}$ (2-, 0- bzw. 1-stellig) und dem 2-stelligen Relationssymbol $=$.

Im folgenden sei die Sprache $\mathcal{L}$ fest gewählt; wir lassen deshalb den Bezug auf $\mathcal{L}$ weg.

Terme werden induktiv wie folgt definiert.

- Jede Variable ist ein Term.
- Jede Konstante ist ein Term.
- Sind $t_1, \ldots, t_n$ Terme und ist f ein n-stelliges Funktionssymbol mit $n \geq 1$, so ist $f(t_1, \ldots, t_n)$ ein Term.

Mit Ter bezeichnen wir die Menge der Terme. Für jeden Term t definieren wir wie folgt die Menge $\mathsf{vars}(t)$ aller Variablen, die in dem Term auftreten.

- Ist t eine Variable, so ist $\mathsf{vars}(t) := \{t\}$.
- Ist t eine Konstante, so ist $\mathsf{vars}(t) := \emptyset$.
- Sind $t_1, \ldots, t_n$ Terme und ist f ein n-stelliges Funktionssymbol, so ist

$$\mathsf{vars}(f(t_1, \ldots, t_n)) := \mathsf{vars}(t_1) \cup \cdots \cup \mathsf{vars}(t_n).$$

Ist $\mathsf{vars}(t) = \emptyset$, so heißt der Term t *geschlossen* oder *Grundterm*.

Zweistellige Funktionssymbole werden meist in *Infix*-Schreibweise verwendet. Man schreibt dann $x + y$ statt $+(x, y)$. Klammern werden so gesetzt, daß sich die Struktur eines Terms eindeutig ergibt. Statt $f(t_1, \ldots, t_n)$ schreiben wir oft auch $ft_1 \ldots t_n$. Terme der Sprache der Gruppentheorie, also $\mathcal{L}_G$-Terme, sind dann zum Beispiel e, $x \circ e$ und $(x^{-1} \circ y^{-1})^{-1}$.

Aus Termen erhält man jetzt sehr einfach *Primformeln* oder *atomare Formeln*: Sind $t_1, \ldots, t_n$ Terme und ist R ein n-stelliges Relationssymbol, so ist $R(t_1, \ldots, t_n)$ eine Primformel; insbesondere ist $\perp$ eine Primformel.

Auch zweistellige Relationssymbole werden meist in *Infix*-Schreibweise verwendet; man schreibt $x < y$ statt $<(x, y)$. Klammern werden so gesetzt, daß sich die Struktur einer atomaren Formel eindeutig ergibt. Statt $R(t_1, \ldots, t_n)$ schreiben wir oft $Rt_1 \ldots t_n$.

Formeln werden aus Primformeln wieder induktiv definiert durch

- Jede Primformel ist eine Formel.
- Sind A und B Formeln, so auch $A \wedge B$ und $A \to B$.
- Ist A eine Formel und x eine Variable, so ist $\forall x A$ eine Formel.

Für jede Formel A definieren wir wie folgt die Menge $\mathsf{FV}(A)$ aller in A *freien Variablen*.

$$\begin{aligned}
\mathsf{FV}(R(t_1, \ldots, t_n)) &:= \mathsf{vars}(t_1) \cup \cdots \cup \mathsf{vars}(t_n),\\
\mathsf{FV}(A \wedge B) &:= \mathsf{FV}(A) \cup \mathsf{FV}(B),\\
\mathsf{FV}(A \to B) &:= \mathsf{FV}(A) \cup \mathsf{FV}(B),\\
\mathsf{FV}(\forall x A) &:= \mathsf{FV}(A) \setminus \{x\}.
\end{aligned}$$

Falls $\mathsf{FV}(A) = \emptyset$, so heißt A *geschlossene Formel* oder *Satz*. Wir schreiben $A[\vec{x}]$, um mitzuteilen, daß alle freien Variablen von A in der Liste $\vec{x}$ enthalten sind.

Wir beschränken uns hier der Einfachheit halber auf die logischen Symbole $\wedge$, $\to$ und $\forall$, da Negation, Disjunktion, Äquivalenz und der Existenzquantor daraus definierbar sind, und zwar durch

$$\begin{aligned}
\neg A &:= A \to \perp,\\
A \vee B &:= \neg(\neg A \wedge \neg B),\\
A \leftrightarrow B &:= (A \to B) \wedge (B \to A),\\
\exists^{\mathsf{cl}} x A &:= \neg \forall x \neg A.
\end{aligned}$$

Wir schreiben $t \neq s$ für $\neg(t = s)$ und $t \not< s$ für $\neg(t < s)$. Klammern werden so gesetzt, daß sich die Struktur einer Formel eindeutig ergibt. Unsere Klammerkonventionen sind wie folgt zu erweitern.

(a) Der Wirkungsbereich von $\neg$ und $\forall x$ ist so klein wie möglich. Beispiel:

$$\forall x A \to B \text{ meint } (\forall x A) \to B, \text{ nicht } \forall x(A \to B).$$

(b) Schreibt man hinter $\forall x$ einen Punkt, so ist der Wirkungsbereich von $\forall x$ ist so groß wie möglich. Beispiel:

$$\forall x.A \to B \text{ meint } \forall x(A \to B), \text{ nicht } (\forall x A) \to B.$$

Als Mitteilungszeichen verwenden wir (auch mit Indizes)

r, s, t	für Terme
x, y, z	für Variablen
c	für Konstanten
P, Q, R	für Relationssymbole
f, g, h	für Funktionssymbole
A, B, C, D	für Formeln

Der Einfachheit halber identifizieren wir Formeln, die sich nur durch gebundene Umbenennung unterscheiden. Dann läßt sich die Substitution $A[x := t]$ eines Terms t für eine Variable x besonders einfach definieren:

Für Terme r und t definieren wir das Ergebnis der *Substitution* von t für x in r wie folgt durch Induktion über den Aufbau von r.

$$\begin{aligned}
y[x := t] &:= \begin{cases} t & \text{falls } x = y \\ y & \text{sonst} \end{cases} \\
c[x := t] &:= c \\
f(t_1, \ldots, t_n)[x := t] &:= f(t_1[x := t], \ldots, t_n[x := t])
\end{aligned}$$

Für Formeln A und Terme t definieren wir das Ergebnis der Substitution von t für x in A wie folgt durch Induktion über den Aufbau von A.

$$\begin{aligned}
R(t_1, \ldots, t_n)[x := t] &:= R(t_1[x := t], \ldots, t_n[x := t]) \\
(A \wedge B)[x := t] &:= A[x := t] \wedge B[x := t] \\
(A \to B)[x := t] &:= A[x := t] \to B[x := t] \\
(\forall y A)[x := t] &:= \forall y A[x := t], \quad \text{wobei oBdA } y \neq x \text{ und } y \notin \mathsf{vars}(t).
\end{aligned}$$

2. Beweise

Wir erweitern jetzt das bisher nur für die Aussagenlogik angegebene System des natürlichen Schließens um *Einführungs- und Beseitigungsregeln* für den Allquantor.

Wir beginnen wieder mit Beispielen für natürliche Beweise.

Gegeben sei also eine Sprache $\mathcal{L}$ erster Stufe. Der Einfachheit halber betrachten wir Beweise in der reinen Logik, d.h. ohne Annahmen über die Funktionen und Relationen.

(3) $$(\forall x.A \to B) \to A \to \forall x B, \quad \text{falls } x \notin \mathsf{FV}(A).$$

Beweis. Gelte $\forall x.A \to B$. Zu zeigen: $A \to \forall x B$. Gelte also A. Zu zeigen: $\forall x B$. Sei x beliebig; man beachte, daß wir bisher keine Annahmen über x gemacht haben. Zu zeigen: B. Wir haben $A \to B$, nach der ersten Annahme. Also auch B, nach der zweiten Annahme. □

(4) $$(A \to \forall x B) \to \forall x.A \to B, \quad \text{falls } x \notin \mathsf{FV}(A).$$

BEWEIS. Gelte $A \to \forall x B$. Zu zeigen: $\forall x.A \to B$. Sei x beliebig; man beachte, daß wir bisher keine Annahmen über x gemacht haben. Zu zeigen: $A \to B$. Gelte also A. Zu zeigen: B. Wir haben $\forall x B$, nach der ersten und zweiten Annahme. Also auch B. $\square$

Auch im Fall der Quantorenlogik können wir Herleitungen als beschriftete Bäume auffassen. Die Beschriftungen der inneren Knoten sind wieder Formeln, die der Blätter sind Formeln oder auch Terme.

Die in der Quantorenlogik neu auftretenden Variablen nennen wir oft auch *Objektvariablen*, um sie von den Annahmenvariablen zu unterscheiden.

Eine Regel kann jetzt nicht nur eine Annahmenvariable binden (und damit die Abhängigkeit von allen darüber stehenden, mit u markierten Annahmen $u\colon A$ beseitigen), sondern auch eine Objektvariable x (und damit die Abhängigkeit von x beseitigen). Dann wird die abgebundene Annahmen- oder Objektvariable der Beschriftung hinzugefügt.

Die Regeln des natürlichen Schließens müssen erweitert werden: Für den Allquantor $\forall$ gibt es eine Einführungsregel $\forall^+ x$ und eine Beseitigungsregel $\forall^-$, die als rechte Prämisse den zu substituierenden Term t hat. Die Regel $\forall^+ x$ unterliegt der folgenden *Variablenbedingung*: Die Herleitung $\mathcal{D}$ der Prämisse A darf keine offenen Annahmen enthalten, in denen x frei vorkommt.

$$\dfrac{\begin{array}{c}\mathcal{D}\\ A\end{array}}{\forall x A}\,\forall^+ x \qquad\qquad \dfrac{\begin{array}{c}\mathcal{D}\\ \forall x A\end{array} \quad t}{A[x := t]}\,\forall^-$$

Wir geben Herleitungen für die oben bewiesenen Formeln (3), (4) an. Da meist die verwendete Regel durch die an den Knoten stehenden Formeln bestimmt ist, verzichten wir im allgemeinen auf die Angabe der Regel.

$$\dfrac{\dfrac{\dfrac{\dfrac{u\colon \forall x.A \to B \quad x}{A \to B} \quad v\colon A}{B}}{\forall x B}\,\forall^+ x}{A \to \forall x B}\,\to^+ v}{(\forall x.A \to B) \to (A \to \forall x B)}\,\to^+ u \tag{3}$$

Hier ist zu beachten, daß die Variablenbedingung erfüllt ist: x kommt nicht frei in A (und auch nicht frei in $\forall x.A \to B$) vor.

$$\dfrac{\dfrac{\dfrac{\dfrac{u\colon A \to \forall x B \quad v\colon A}{\forall x B} \quad x}{B}}{A \to B}\,\to^+ v}{\forall x.A \to B}\,\forall^+ x}{(A \to \forall x B) \to \forall x.A \to B}\,\to^+ u \tag{4}$$

Auch hier ist die Variablenbedingung erfüllt: x kommt nicht frei in A vor.

Lineare Formulierung des natürlichen Schließens. Die schon im Fall der Aussagenlogik zur Behandlung der abzubindenden Annahmenvariablen in $\to^+$ erforderliche *Blockstruktur* muß jetzt für die Regel $\forall^+$ erweitert werden.

Ein *Block* beginnt wieder mit { und endet mit }. Nach der öffnenden geschweiften Klammer kann jetzt auch eine Variable folgen, die damit lokal

für den Block festgelegt wird. Nach dem Schließen des Blocks ist dann eine Allformel bewiesen. Natürlich ist hier die Variablenbedingung zu überprüfen.

Für unser obiges Beispiel (3) schreibt man

```
MPC;
alpha . x;
PRED . P;
PRED alpha . Q;
PROOF;
{ all x.P -> Q x.
  { P.
    { x.
      P -> Q x;
      Q x;
    }
  }
}
```

Eine solche Textdatei kann man wieder vom Beweisprüfer MPC überprüfen lassen, mit folgendem Ergebnis:

```
{ all x.P -> Q x.       // 0 assumed.
  { P.  // 1 assumed.
    { x.        // x assumed.
      P -> Q x; // ok, 2 proved by all-elim from 0
      Q x;      // ok, 3 proved by imp-elim from 2 and 1
    }   // ok, 4 all x Q x proved.
  }     // ok, 5 P -> all x Q x proved.
}       // ok, 6 (all x.P -> Q x) -> P -> all x Q x proved.
```

Im Beispiel (4) gibt man ein

```
MPC;
alpha . x;
PRED . P;
PRED alpha . Q;
PROOF;
{ P -> all x Q x.
  { x.
    { P.
      all x Q x;
      Q x;
    }
  }
}
```

und erhält als Kommentare

```
{ P -> all x Q x.       // 0 assumed.
  { x.  // x assumed.
    { P.        // 1 assumed.
      all x Q x;  // ok, 2 proved by imp-elim from 0 and 1
      Q x;      // ok, 3 proved by all-elim from 2
    }   // ok, 4 P -> Q x proved.
```

```
  }     // ok, 5 all x.P -> Q x proved.
}       // ok, 6 (P -> all x Q x) -> all x.P -> Q x proved.
```

3. Intuitionistische und klassische Logik

In der Quantorenlogik müssen die *Ex-Falso-Quodlibet*-Formeln (oder „Axiome“) etwas allgemeiner formuliert werden. Wir benötigen jetzt Efq_R für jedes von $\bot$ verschiedene Relationssymbol R:

$$\forall \vec{x}.\bot \to R\vec{x} \qquad (\mathsf{Efq}_R)$$

sowie — für die klassische Logik — als Prinzip des indirekten Beweisens für jedes von $\bot$ verschiedene Relationssymbol R die Formel

$$\forall \vec{x}.\neg\neg R\vec{x} \to R\vec{x}; \qquad (\mathsf{Stab}_R)$$

diese Formel bezeichnet man als *Stabilität* von R.

Sei

$$\begin{aligned} \mathsf{Efq} &:= \{\, \mathsf{Efq}_R \mid R \text{ Relationssymbol} \neq \bot \,\}, \\ \mathsf{Stab} &:= \{\, \mathsf{Stab}_R \mid R \text{ Relationssymbol} \neq \bot \,\}. \end{aligned}$$

Die Begriffe $\vdash_c A$ und $\vdash_i A$ der klassischen und intuitionistische Herleitbarkeit übertragen sich in offensichtlicher Weise. Die Lemmata sind ebenfalls leicht zu erweitern:

Lemma 3.1 (Ex-falso-quodlibet). *Für jede Formel A gilt $\vdash_i \bot \to A$.*

Beweis. Durch Induktion über A konstruieren wir für jede Formel A eine Herleitung $\mathcal{D}_A$ von $\bot \to A$. *Fall* $\forall x A$.

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{D}_A\\ \bot \to A\end{array} \qquad u\colon \bot}{A}}{\forall x A}}{\bot \to \forall x A}\ \to^+ u$$

Damit ist der Beweis abgeschlossen. □

Lemma 3.2 (Stabilität). *Für jede Formel A (unserer $\to\wedge\forall$-Sprache) gilt $\vdash_c \neg\neg A \to A$.*

Beweis. Induktion über A. *Fall* $\forall x A$. Offenbar genügt es zu zeigen, daß $\vdash (\neg\neg A \to A) \to \neg\neg\forall x A \to A$. Eine Herleitung ist

$$\dfrac{u\colon \neg\neg A \to A \qquad \dfrac{\dfrac{v\colon \neg\neg\forall x A \qquad \dfrac{\dfrac{u_1\colon \neg A \qquad \dfrac{u_2\colon \forall x A \qquad x}{A}}{\bot}}{\neg\forall x A}\ \to^+ u_2}{\bot}}{\neg\neg A}\ \to^+ u_1}{A}$$

Damit ist der Beweis abgeschlossen. □

Lemma 3.3. $\Gamma \vdash A \Rightarrow \Gamma \vdash_i A$ *und* $\Gamma \vdash_i A \Rightarrow \Gamma \vdash_c A$.

Beweis. Es genügt zu zeigen, daß $\vdash_c \mathsf{Efq}_R$. Dies sieht man wie folgt; R sei etwa einstellig.

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall x.\neg\neg Rx \to Rx \qquad x}{\neg\neg Rx \to Rx} \qquad \dfrac{u\colon \bot}{\neg\neg Rx}\to^+ v^{\neg Rx}}{Rx}}{\bot \to Rx}\to^+ u}{\forall x.\bot \to Rx}\forall^+$$

Damit ist der Beweis abgeschlossen. □

Auch der Beweis des Äquivalenzlemmas läßt sich leicht auf die Quantorenlogik erweitern:

Lemma 3.4 (Äquivalenzlemma). *Für* $\vdash_{mic} \in \{\vdash, \vdash_i, \vdash_c\}$ *gilt folgendes. Ist* $\vdash_{mic} A_1 \leftrightarrow A_2$ *und entsteht* B_2 *aus* B_1 *durch Ersetzen eines Teils* A_1 *von* B_1 *durch* A_2*, so gilt auch* $\vdash_{mic} B_1 \leftrightarrow B_2$.

Beweis. Induktion über B_1. Falls ganz B_1 ersetzt wird, so ist die Behauptung klar. Andernfalls muß B_1 eine zusammengesetzte Formel sein.

Fall $\forall x C_1$. Zu zeigen ist $\vdash_{mic} \forall x C_1 \leftrightarrow \forall x C_2$. $\to$:

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}\\ C_1 \to C_2\end{matrix} \qquad \dfrac{\forall x C_1 \qquad x}{C_1}}{C_2}}{\forall x C_2}$$

wobei wieder $\mathcal{D}$ nach IH bekannt ist. Man beachte, daß $\mathcal{D}$ keine freien Annahmen enthält. $\leftarrow$ beweist man ähnlich. □

Einbettung der intuitionistischen und klassischen Logik in die Minimallogik. Auch hierfür macht die Erweiterung auf den Allquantor keine Probleme. Die Definition einer negativen Formel bleibt unverändert: eine Formel heißt *negativ*, wenn jede atomare Formel $\neq \bot$ in A negiert vorkommt, d.h. in einem Kontext $R\vec{t} \to \bot$.

Lemma 3.5. *Für negative* A *gilt* $\vdash \neg\neg A \to A$.

Beweis. Dies beweist man wie das Stabilitätslemma 3.2 durch Induktion über A; man verwendet $\vdash \neg\neg\neg R\vec{t} \to \neg R\vec{t}$ anstelle der Stabilitätsannahmen. □

Die negative Übersetzung ^{g} nach Gödel-Gentzen erweitern wir durch die zusätzliche Klausel

$$(\forall x A)^g := \forall x A^g.$$

Ihre wesentlichen Eigenschaften, also

(a) $\vdash_c A \leftrightarrow A^g$,
(b) $\Gamma \vdash_c A$ genau dann, wenn $\Gamma^g \vdash A^g$, wobei $\Gamma^g := \{\, B^g \mid B \in \Gamma \,\}$.

bleiben (mit demselben Beweis) erhalten.

Dasselbe gilt für das Korollar über die Einbettung der klassischen Logik in die Minimallogik.

Abgeleitete Regeln für den klassischen Existenzquantor. Den klassischen Existenzquantor hatten wir definiert durch

$$\exists^{\mathsf{cl}} x A := \neg\forall x \neg A.$$

Herleitungen von Formeln mit $\exists^{\mathsf{cl}}$ lassen sich jedoch oft leichter finden, wenn man nicht auf diese Definitionen zurückgeht, sondern direkt mit „*abgeleiteten Regeln*“ für $\exists^{\mathsf{cl}}$ arbeitet. Darunter verstehen wir die folgenden Regeln. Es gibt eine Einführungsregel $\exists^{\mathsf{cl}+}$, die als rechte Prämisse den zu substituierenden Term t hat, und eine Beseitigungsregel $\exists^{\mathsf{cl}-} u$. Die Regel $\exists^{\mathsf{cl}-} u$ unterliegt der folgenden *Variablenbedingung*: Die Herleitung $\mathcal{D}_0$ darf keine offenen Annahmen außer $u\colon A$ enthalten, in denen x frei vorkommt, und ferner darf B die Variable x nicht frei enthalten.

$$\dfrac{\begin{array}{c}\mathcal{D}\\ A[x := t]\end{array} \qquad t}{\exists^{\mathsf{cl}} x A}\,\exists^{\mathsf{cl}+} \qquad\qquad \dfrac{\begin{array}{c}\mathcal{D}\\ \exists^{\mathsf{cl}} x A\end{array} \qquad \begin{array}{c}[u\colon A]\\ \mathcal{D}_0\\ B\end{array}}{B}\,\exists^{\mathsf{cl}-} u$$

Der doppelte Strich deutet wieder an, daß es sich nicht um eine primitive Regel unserer Logik handelt, sondern daß man an dieser Stelle einen Herleitungsteil so einsetzen kann, daß eine korrekt gebildete Herleitung entsteht. Wir geben im folgenden für jede der abgeleiteten Regeln den einzusetzenden Herleitungsteil an. Man beachte, daß im Fall der Beseitigungsregel $\exists^{\mathsf{cl}-} u$ eine Stabilitätsannahme verwendet wird, man also klassische Logik benötigt.

$\exists^{\mathsf{cl}+}$:

$$\dfrac{\dfrac{\dfrac{u\colon \forall x \neg A \qquad t}{\neg A[x := t]} \qquad \begin{array}{c}\mathcal{D}\\ A[x := t]\end{array}}{\bot}}{\neg\forall x \neg A}\,\to^{+} u$$

$\exists^{\mathsf{cl}-} u$:

$$\dfrac{\begin{array}{c}\mathcal{D}_{\mathsf{Stab}}\\ \neg\neg B \to B\end{array} \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ \neg\forall x \neg A\end{array} \qquad \dfrac{\dfrac{v\colon \neg B \qquad \begin{array}{c}[u\colon A]\\ \mathcal{D}_0\\ B\end{array}}{\bot}}{\neg A}\,\to^{+} u}{\forall x \neg A}}{\dfrac{\bot}{\neg\neg B}\,\to^{+} v}}{B}$$

Als Alternative kann man ebenfalls den Komfort der abgeleiteten Regeln für $\exists^{\mathsf{cl}}$ erhalten, indem man die folgenden (ableitbaren) Schemata als Axiome benutzt.

$$A \to \exists^{\mathsf{cl}} x A,$$

$$\exists^{\mathsf{cl}} x A \to (\forall x. A \to B) \to B, \quad \text{falls } x \notin \mathsf{FV}(B).$$

4. Konjunktive, disjunktive und pränexe Normalformen

Zur Erweiterung der konjunktiven und disjunktiven Normalformen benötigen wir den Begriff der *Quantorentiefe* $\mathsf{qt}(A)$ einer Formel A; er ist durch

Rekursion über A definiert wie folgt.

$$\begin{aligned}
\mathsf{qt}(R\vec{t}) &:= 0,\\
\mathsf{qt}(A \circ B) &:= \max(\mathsf{qt}(A), \mathsf{qt}(B)) \quad \text{für } \circ \in \{\wedge, \to\},\\
\mathsf{qt}(\forall x A) &:= \mathsf{qt}(A) + 1.
\end{aligned}$$

SATZ 4.1 (Konjunktive und disjunktive Normalform). *Für jede Formel A der Quantorenlogik gilt folgendes.*

(a) *A ist klassisch äquivalent zu einer Formel der Gestalt*

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} A_{ij} \qquad (\text{konjunktive Normalform}),$$

wobei die A_{ij} aussagenlogisch unzerlegbare Formeln (d.h. atomare Formeln oder von der Form $\forall x A$) oder Negationen davon sind, und wobei gilt $\mathsf{qt}(A_{ij}) \leq \mathsf{qt}(A)$.

(b) *A ist klassisch äquivalent zu einer Formel der Gestalt*

$$\bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} A_{ij} \qquad (\text{disjunktive Normalform}),$$

wobei wieder die A_{ij} aussagenlogisch unzerlegbare Formeln oder Negationen davon sind, und $\mathsf{qt}(A_{ij}) \leq \mathsf{qt}(A)$.

BEWEIS. Wie in Kapitel 1. □

Zusätzlich zur der konjunktiven und disjunktiven Normalform gibt es in der klassischen Quantorenlogik noch die Möglichkeit, Quantoren nach außen zu ziehen. Dies führt auf die *pränexe Normalform.*

LEMMA 4.2 (Herausziehen von Quantoren). *Sei $x \notin \mathsf{FV}(A)$. Dann gilt*

(a) $\vdash_c A \wedge \forall x B \leftrightarrow \forall x.A \wedge B$.
(b) $\vdash_c A \wedge \exists^{\mathsf{cl}} x B \leftrightarrow \exists^{\mathsf{cl}} x.A \wedge B$.
(c) $\vdash_c (A \to \forall x B) \leftrightarrow \forall x.A \to B$.
(d) $\vdash_c (A \to \exists^{\mathsf{cl}} x B) \leftrightarrow \exists^{\mathsf{cl}} x.A \to B$.
(e) $\vdash_c (\forall x B \to A) \leftrightarrow \exists^{\mathsf{cl}} x.B \to A$.
(f) $\vdash_c (\exists^{\mathsf{cl}} x B \to A) \leftrightarrow \forall x.B \to A$.

BEWEIS. Wir behandeln hier und im folgenden die abgeleiteten Regeln für $\exists^{\mathsf{cl}}$ (und $\vee$) wie gewöhnliche Regeln.

(a) ist sehr leicht zu zeigen. (b) $\to$.

$$\dfrac{v\colon \exists^{\mathsf{cl}} x B \qquad \dfrac{\dfrac{u\colon A \qquad u_0\colon B}{A \wedge B} \qquad x}{\exists^{\mathsf{cl}} x.A \wedge B}}{\exists^{\mathsf{cl}} x.A \wedge B}\ \exists^{\mathsf{cl}-} u_0$$

$\leftarrow$.

$$\dfrac{\dfrac{u\colon \exists^{\mathsf{cl}} x.A \wedge B \qquad \dfrac{u_0\colon A \wedge B}{A}}{A}\ \exists^{\mathsf{cl}-} u_0 \qquad \dfrac{u\colon \exists^{\mathsf{cl}} x.A \wedge B \qquad \dfrac{\dfrac{u_1\colon A \wedge B}{B} \qquad x}{\exists^{\mathsf{cl}} x B}}{\exists^{\mathsf{cl}} x B}\ \exists^{\mathsf{cl}-} u_1}{A \wedge \exists^{\mathsf{cl}} x B}$$

(c) Dies hatten wir in (3) und (4) gezeigt.

(d) $\to$. Hier helfen die abgeleiteten Regeln für $\exists^{\mathsf{cl}}$ nicht; wir müssen auf die Definition von $\exists^{\mathsf{cl}}$ zurückgehen. Ferner ist es nützlich, sich an die (klassische) Äquivalenz von $\neg(A \to B)$ mit $A \wedge \neg B$ zu erinnern. Wir führen den Beweis informal, und zwar so, daß klar ist, wie man daraus (unter Zuhilfenahme des Äquivalenzlemmas 3.5) eine formale Herleitung gewinnen kann. Gelte also (*) $A \to \neg\forall x \neg B$ und (**) $\forall x.A \wedge \neg B$. Aus (**) erhält man A, also aus (*) $\neg\forall x\neg B$. Aus (**) erhält man auch $\forall x \neg B$, also insgesamt $\bot$, wie gewünscht.

(d) $\leftarrow$.

$$\dfrac{\dfrac{u\colon \exists^{\mathsf{cl}}x.A \to B \qquad \dfrac{\dfrac{u_0\colon A \to B \qquad u_1\colon A}{B} \qquad x}{\exists^{\mathsf{cl}}xB}\,\exists^{\mathsf{cl}-}u_0}{\exists^{\mathsf{cl}}xB}}{A \to \exists^{\mathsf{cl}}xB}\,\to^{+}u_1$$

(e) und (f): Übung. □

Eine Formel A heißt *pränex*, wenn $A = Q_1x_1Q_2x_2\ldots Q_nx_nB$ mit $Q_i \in \{\forall, \exists^{\mathsf{cl}}\}$ und B quantorenfrei. Die Formel B heißt der *Kern* von A und $Q_1x_1Q_2x_2\ldots Q_nx_n$ das *Präfix* von A. Wir zeigen jetzt, daß sich jede Formel in eine *pränexe Normalform* bringen läßt.

Satz 4.3 (Pränexe Normalform). *Zu jeder Formel A findet man eine klassisch äquivalente pränexe Formel A'.*

Beweis. Induktion über A.

Fall atomare Formel. In diesem Fall ist nichts zu zeigen.

Fall $A \wedge B$. Nach IH haben wir $\vdash_c A \leftrightarrow A'$ und $\vdash_c B \leftrightarrow B'$ mit

$$A' = Q_1x_1\ldots Q_nx_nA_0, \quad A_0 \text{ quantorenfrei},$$
$$B' = Q_1^*y_1\ldots Q_m^*y_mB_0, \quad B_0 \text{ quantorenfrei}.$$

Wir können oBdA annehmen, daß kein x_i in B' und kein y_j in A' vorkommt. Nach dem Lemma 4.2 über das Herausziehen von Quantoren und dem Äquivalenzlemma 3.5 folgt

$$\vdash_c A \wedge B \leftrightarrow Q_1x_1\ldots Q_nx_nQ_1^*y_1\ldots Q_m^*y_m.A_0 \wedge B_0.$$

Fall $A \to B$. Wie eben erhält man

$$\vdash_c (A \to B) \leftrightarrow \overline{Q}_1x_1\ldots\overline{Q}_nx_nQ_1^*y_1\ldots Q_m^*y_m.A_0 \wedge B_0.$$

wobei $\overline{Q}_i = \forall$ falls $Q_i = \exists^{\mathsf{cl}}$, und $\overline{Q}_i = \exists^{\mathsf{cl}}$ falls $Q_i = \forall$.

Fall $\forall xA$. Nach IH haben wir $\vdash_c A \leftrightarrow A'$ mit einem pränexen A'. Nach dem Äquivalenzlemma 3.5 folgt

$$\vdash_c \forall xA \leftrightarrow \forall xA'.$$

Das war zu zeigen. □

Zur Berechnung der pränexen Normalform ist es nützlich, die folgenden leicht zu beweisenden Äquivalenzen zu verwenden.

Lemma 4.4. (a) $\vdash_c \neg\forall xB \leftrightarrow \exists^{\mathsf{cl}}x\neg B$.
(b) $\vdash_c \neg\exists^{\mathsf{cl}}xB \leftrightarrow \forall x\neg B$.
(c) $\vdash_c A \vee \forall xB \leftrightarrow \forall x.A \vee B$, *falls x notin*$\mathsf{FV}(A)$.

(d) $\vdash_c A \vee \exists^{\mathsf{cl}} x B \leftrightarrow \exists^{\mathsf{cl}} x.A \vee B$, *falls* $x \notin \mathsf{FV}(A)$.

BEWEIS. Übung. □

5. Starke Disjunktion und Existenz

In der Mathematik ist es möglich und sinnvoll, zwischen Existenzbeweisen zu unterscheiden, die das als existent nachgewiesene Objekt tatsächlich liefern, und solchen, die dies nicht tun. Als Beispiel betrachten wir die folgende Aussage.

Es gibt irrationale Zahlen a, b mit a^b rational.

Einen Beweis erhält man wie folgt durch Fallunterscheidung.

Fall $\sqrt{2}^{\sqrt{2}}$ ist rational. Man wähle $a = \sqrt{2}$ und $b = \sqrt{2}$. Dann sind a, b irrational, und nach Annahme ist a^b rational.

Fall $\sqrt{2}^{\sqrt{2}}$ ist irrational. Man wähle $a = \sqrt{2}^{\sqrt{2}}$ und $b = \sqrt{2}$. Dann sind nach Annahme a, b irrational, und

$$a^b = \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \left(\sqrt{2}\right)^2 = 2$$

ist rational. □

Solange wir nicht entschieden haben, ob $\sqrt{2}^{\sqrt{2}}$ nun rational ist oder nicht, wissen wir nicht, welche Zahlen a, b wir nehmen müssen. Damit haben wir ein Beispiel eines Existenzbeweises, der es nicht erlaubt, das als existent nachgewiesene Objekt tatsächlich anzugeben.

Neben dem bisher behandelten *schwachen* oder *klassischen* Existenzquantor $\exists^{\mathsf{cl}}$ (der durch $\neg\forall\neg$ definiert war) wollen wir deshalb auch noch einen *starken* oder *konstruktiven* Existenzquantor $\exists$ zulassen. Man kann dann in den Fällen, wo ein Existenzbeweis tatsächlich konstruktiv durch Angabe eines Beispiels geführt wurde, dies auch in der Formelsprache angemessen ausdrücken.

Entsprechend könnte man auch neben der bisher behandelten *schwachen* oder *klassischen* Disjunktion $\vee$ (die durch $\neg \wedge \neg$ definiert war) auch noch eine *starke* oder *konstruktive* Disjunktion $\vee^*$ zulassen. In Anwesenheit des Grundtyps der natürlichen Zahlen ist dies jedoch entbehrlich: Wir definieren

$$A \vee^* B \quad := \quad \exists n.(n = 0 \to A) \wedge (n \neq 0 \to B).$$

Wir wollen kurz diskutieren, welchen Effekt die Hinzunahme von $\vee^*, \exists$ auf unsere bisherigen Untersuchungen hat. Wir erweitern also den Formelbegriff um die Klauseln $A \vee^* B$ und $\exists x A$. Die Definitionen der Mengen der freien Variablen und der Substitution werden in der offensichtlichen Weise erweitert, ebenso unsere Konventionen betreffend Klammersetzung. Die Logik von $\vee^*$ und $\exists$ kann dann durch die abgeleiteten Regeln für den Existenzquantor und die Disjunktion axiomatisiert werden. Alternativ kann man auch die Regeln der Minimallogik beibehalten und den starken Existenzquantor durch die Generalisierung der folgenden Formeln axiomatisieren; diesen Weg wollen wir hier gehen.

$$A \to A \vee^* B, \quad B \to A \vee^* B.$$
$$A \vee^* B \to (A \to C) \to (B \to C) \to C.$$

$$A \to \exists x A.$$
$$\exists x A \to (\forall x.A \to B) \to B, \quad \text{falls } x \notin \mathsf{FV}(B).$$

Diese Axiomenschemata bezeichnen wir durch $\vee_0^{*+}$, $\vee_1^{*+}$, $\vee^{*-}$, $\exists^+$ und $\exists^-$. Für Formeln A in der durch $\vee^*$ und $\exists$ erweiterten Sprache schreiben wir $\vdash A$ und nennen A *herleitbar* (in der *Minimallogik*), wenn es eine Herleitung von A aus diesen Axiomenschemata gibt.

LEMMA 5.1 (Ex-falso-quodlibet). *Für jede Formel A in der Sprache mit den Verknüpfungen $\to$, $\wedge$, $\vee^*$, $\forall$ und $\exists$ gilt*

$$\vdash_i \bot \to A.$$

BEWEIS. Wir müssen nur die Fälle $A \vee^* B$ und $\exists x A$ zusätzlich behandeln, was aber trivial ist. □

Die Einbettung der intuitionistischen Logik läßt sich also wie bisher durchführen.

Lineare Formulierung des natürlichen Schließens mit dem starken Existenzquantor. Ein Beweis einer Formel B kann eine existentiell quantifizierte Formel $\exists x\, A$ verwenden. Wie bereits diskutiert, verläuft ein solcher Beweis üblicherweise wie folgt. Wir wissen, daß es ein x mit A gibt. Nehmen wir also an, wir hätten ein solches x. Nennen wir es wieder x, so können wir A annehmen. Unter dieser Annahme beweisen wir B. Dies läßt sich als ein Beweis von B schlechthin auffassen, falls die Variablenbedingung erfüllt ist: weder in B noch in einer noch freien Annahme außer der erwähnten Annahme A darf x frei vorkommen.

Dieser Beweis kann in MPC genau wie eben beschrieben mit Hilfe von zwei geschachtelten Blöcken formuliert werden. Der erste Block führt die lokale Variable x und der zweite Block die lokale Annahme A ein. Sobald die Formel B hieraus bewiesen ist, werden beide Blöcke geschlossen. Damit ist dann die Formel $\forall x.A \to B$ bewiesen. Dies, zusammen mit der Formel $\exists x\, A$, kann man dann verwenden um mit Hilfe der Regel der Existenzbeseitigung auf B zu schließen. Als Beispiel beweisen wir $(\exists x.Q_1 x \wedge Q_2 x) \to \exists x Q_1 x$.

```
MPC;
alpha . x;
PRED alpha . Q;
PROOF;
{ ex x.Q1 x & Q2 x.
  { x.
    { Q1 x & Q2 x.
      Q1 x;
      ex x Q1 x;
    }
  }
  ex x Q1 x;
}
```

Schreibt man dies wieder in eine Textdatei und überprüft diese durch den Beweisprüfer MPC, so erhält man:

```
{ ex x.Q1 x & Q2 x.     // 0 assumed.
  { x.  // x assumed.
    { Q1 x & Q2 x.      // 1 assumed.
      Q1 x;     // ok, 2 proved by and-elim-left from 1
      ex x Q1 x;        // ok, 3 proved by ex-intro from 2
    }   // ok, 4 Q1 x & Q2 x -> ex x Q1 x proved.
  }     // ok, 5 all x.Q1 x & Q2 x -> ex x Q1 x proved.
  ex x Q1 x;    // ok, 6 proved by ex-elim from 5 and 0
}       // ok, 7 (ex x.Q1 x & Q2 x) -> ex x Q1 x proved.
```

6. Rechnergestütztes Beweisen

Wir haben bisher Rechnerunterstützung nur in dem bescheidenen Sinn verwendet, daß wir einen Beweisprüfer verwendet haben. Lästing war dabei, daß

- die Formeln explizit eingegeben werden mußten, und
- immer nur einzelne Elementarschritte durchgeführt werden konnten.

Offensichtlich sollte es wegen der formalen Natur der logischen Schlüsse aber leicht möglich sein, eine erheblich weiter gehende Rechnerunterstützung zu erhalten. Derartige Systeme sind von vielen Forschergruppen in der Welt entwickelt worden. Einige der derzeit wohl bedeutendsten Systeme sind Coq (Coquand und Huet, Paris), Isabelle (Paulson, Cambridge und Nipkow, TU München) und Nuprl (Constable, Cornell). Wir wollen hier ein lokal entwickeltes System (Minlog) verwenden, das für die Zwecke dieser Vorlesung besonders geeignet ist.

Worum geht es? Wir wollen den Rechner veranlassen, aufgrund von möglichst knappen Eingaben einen beabsichtigten Beweis zu schrittweise zu konstruieren. Dabei sollen sich die Eingaben möglichst eng an dem zugrunde liegenden informalen Beweis orientieren. Wie dies gemeint ist, sieht man am besten an Beispielen.

Betrachten wir die Formel

$$(A \to B \to C) \to (A \to B) \to A \to C.$$

Informaler Beweis. Nehmen wir die drei Prämissen 1: $A \to B \to C$, 2: $A \to B$ und 3: A an. Zu zeigen ist C. Wir wollen 1 verwenden, haben also als neue Ziele die Prämissen von 1 zu zeigen, also A und B. Das erste dieser Ziele, also A, ist unmittelbar als Annahme 3 vorhanden. Für das zweite Ziel, also B, verwenden wir die Annahme 2. Wir haben also als neues Ziel die Prämisse von 2 zu zeigen. Dies ist A, was wieder unmittelbar als Annahme 3 vorhanden ist. □

Interaktiver Beweis. Wir beginnen die Beweiskonstruktion mit einem Aufruf des Systems, der Deklaration der Aussagensymbole als Prädikatenkonstanten und der Eingabe unserer Zielformel.

```
> (load "~/minlog/init.scm")
(add-predconst-name "A" "B" "C" (make-arity))
(set-goal (pf "(A -> B -> C) -> (A -> B) -> A -> C"))
```

Die Antwort ist

```
; ok, predicate constant A: (arity) added
; ok, predicate constant B: (arity) added
; ok, predicate constant C: (arity) added
> ; ?_1: (A -> B -> C) -> (A -> B) -> A -> C
```

Als nächstes nehmen wir die drei Prämissen an:

```
(assume 1 2 3)
```

Die Antwort ist

```
; ok, we now have the new goal
; ?_2: C from
;   1:A -> B -> C
;   2:A -> B
;   3:A
```

Als nächstes wollen 1 verwenden, haben also als neue Ziele die Prämissen von 1 zu zeigen:

```
(use-with 1 "?" "?")
```

Die Antwort ist

```
; ok, ?_2 can be obtained from
; ?_4: B from
;   1:A -> B -> C
;   2:A -> B
;   3:A

; ?_3: A from
;   1:A -> B -> C
;   2:A -> B
;   3:A
```

Das oben auf dem Zielstapel liegende Ziel A ist unmittelbar als Annahme 3 vorhanden, die wir nur verwenden müssen:

```
(use 3)
```

Die Antwort ist

```
; ok, ?_3 is proved.  The active goal now is
; ?_4: B from
;   1:A -> B -> C
;   2:A -> B
;   3:A
```

Als nächstes wollen 2 verwenden, haben also als neues Ziel die Prämisse von 2 zu zeigen:

```
(use-with 2 "?")
```

Die Antwort ist

```
; ok, ?_4 can be obtained from
; ?_5: A from
;   1:A -> B -> C
;   2:A -> B
;   3:A
```

Das jetzt oben auf dem Zielstapel liegende Ziel A ist wieder unmittelbar als Annahme 3 vorhanden, die wir nur verwenden müssen:

```
(use 3)
```

Die Antwort ist schließlich

```
; ok, ?_5 is proved.  Proof finished.
```

□

Der hiermit konstruierte Beweis ist im Speicher vorhanden und kann (unabhängig von den vorherigen Konstruktionsschritten) überprüft und in linearer Form ausgegeben werden. Das zugehörige Kommando ist `(cdp)` (für „check and display proof“):

```
; .....A -> B -> C by assumption u13
; .....A by assumption u15
; ....B -> C by imp elim
; .....A -> B by assumption u14
; .....A by assumption u15
; ....B by imp elim
; ...C by imp elim
; ..A -> C by imp intro u15
; .(A -> B) -> A -> C by imp intro u14
; (A -> B -> C) -> (A -> B) -> A -> C by imp intro u13
```

Bemerkung. Es ist offensichtlich, daß die verwendeten Beweisschritte in diesem sehr einfachen Beispiel im Wesentlichen eindeutig bestimmt waren. Es ist deshalb möglich, hier automatisch vorzugehen; das zugehörige Kommando ist `(proceed)`.

Auch wenn diese Eindeutigkeit der Beweisschritte nicht mehr gegeben ist, kann man offenbar systematisch nach einem Beweis suchen. Im Fall der Aussagenlogik (der minimalen, der intuitionistischen und der klassischen) bricht ein solches Suchverfahren stets ab und liefert als Ergebnis entweder einen Beweis oder eine Meldung, daß es keinen Beweis gibt; das zugehörige Kommando ist `(prop)`.

Als weiteres Beispiel betrachten wir eine Formel mit Quantoren:

$$(\forall x, y.Rxy \to Ryx) \to (\forall x, y, z.Rxy \to Ryz \to Rxz) \to \forall x, y.Rxy \to Rxx.$$

Informaler Beweis. Nehmen wir an

$$\texttt{Symm}\colon \forall x, y.Rxy \to Ryx$$
$$\texttt{Trans}\colon \forall x, y, z.Rxy \to Ryz \to Rxz,$$

und ferner beliebige x und y mit `R-Hyp`: Rxy. Zu zeigen ist Rxx. Wir wollen die Annahme `Trans` verwenden, und zwar mit den Argumenten x, y, x, `R-Hyp` und ?. Dies liefert offenbar die gewünschte Zielformel, produziert aber

als neues Ziele Ryx. Wir können jetzt die Annahme `Symm` verwenden, und zwar mit den Argumenten x, y und `R-Hyp`. □

Interaktiver Beweis. Nach dem Aufruf des Systems deklarieren wir die Variablen x, y, z und die Relationskonstante R, und geben die Zielformel ein:

```
> (load "~/minlog/init.scm")
(add-var-name "x" "y" "z" (py "alpha"))
(add-predconst-name "R"
  (make-arity (py "alpha") (py "alpha")))
(set-goal (pf "(all x,y.R x y -> R y x) ->
  (all x,y,z.R x y -> R y z -> R x z) ->
  (all x,y.R x y -> R x x)"))
```

Die Antwort ist

```
; ok, variable x: alpha added
; ok, variable y: alpha added
; ok, variable z: alpha added
> ; ok, predicate constant R: (arity alpha alpha) added
> ; ?_1: (all x,y.R x y -> R y x) ->
         (all x,y,z.R x y -> R y z -> R x z) ->
         all x,y.R x y -> R x x
```

Als nächstes nehmen wir die Prämissen und die allquantifizierten Variablen an:

```
(assume "Symm" "Trans" "x" "y" "R-Hyp")
```

Die Antwort ist

```
; ok, we now have the new goal
; ?_2: R x x from
;   Symm:all x,y.R x y -> R y x
;   Trans:all x,y,z.R x y -> R y z -> R x z
;   x  y  R-Hyp:R x y
```

Wir wollen die Annahme `Trans` verwenden, und zwar mit den Argumenten x, y, x und `R-Hyp` und `?` für das neu zu erzeugende Ziel Ryx:

```
(use-with "Trans" (pt "x") (pt "y") (pt "x") "R-Hyp" "?")
```

Die Antwort ist

```
; ok, ?_2 can be obtained from
; ?_3: R y x from
;   Symm:all x,y.R x y -> R y x
;   Trans:all x,y,z.R x y -> R y z -> R x z
;   x  y  R-Hyp:R x y
```

Wir wollen jetzt die Annahme `Symm` verwenden, und zwar mit den Argumenten x, y und `R-Hyp`:

```
(use-with "Symm" (pt "x") (pt "y") "R-Hyp")
```

Die Antwort ist

```
; ok, ?_3 is proved.  Proof finished.
```

□

Der hiermit konstruierte Beweis kann wieder mit `(cdp)` überprüft und in linearer Form ausgegeben werden. Das Ergebnis (ohne die Einführungen am Schluß) ist:

```
; .....all x,y,z.R x y -> R y z -> R x z by assumption Trans14
; .....x
; ....all y,z.R x y -> R y z -> R x z by all elim
; ....y
; ...all z.R x y -> R y z -> R x z by all elim
; ...x
; ..R x y -> R y x -> R x x by all elim
; ..R x y by assumption R-Hyp15
; .R y x -> R x x by imp elim
; ....all x,y.R x y -> R y x by assumption Symm13
; ....x
; ...all y.R x y -> R y x by all elim
; ...y
; ..R x y -> R y x by all elim
; ..R x y by assumption R-Hyp15
; .R y x by imp elim
; R x x by imp elim
```

Bemerkung. Man kann auch im Fall der minimalen Quantorenlogik systematisch nach einem Beweis suchen. Allerdings bricht ein solches Suchverfahren im allgemeinen nicht ab (da die Beweisbarkeit in der minimalen Quantorenlogik unentscheidbar ist) und liefert nur in günstigen Fällen als Ergebnis einen Beweis; das zugehörige Kommando ist `(search)`. In der Implementierung wird die Terminierung durch künstliche Beschränkung des Suchbaums sichergestellt; man enthält eine entsprechende Meldung, wenn innerhalb dieser Schranke kein Beweis gefunden wurde.

Als letztes Beispiel wollen wir noch die Behandlung des (starken) Existenzquantors besprechen, genauer die oben mit MPC bewiesene Formel

$$(\exists x.Q_1 x \wedge Q_2 x) \to \exists x Q_1 x.$$

Wir beginnen wie immer mit den Deklarationen und dem Setzen des Ziels:

```
(add-var-name "x" (py "alpha"))
(add-predconst-name "Q" (make-arity (py "alpha")))
(set-goal (pf "(ex x.Q1 x & Q2 x) -> ex x Q1 x"))
```

Die Antwort ist

```
; ok, variable x: alpha added
; ok, predicate constant Q: (arity alpha) added
; ?_1: (ex x.Q1 x & Q2 x) -> ex x Q1 x
```

Als nächstes befördern wir die Existenzannahme in den Kontext:

```
(assume "Ex-Hyp")
```

Die Antwort ist

```
; ok, we now have the new goal
; ?_2: ex x Q1 x from
;   Ex-Hyp:ex x.Q1 x & Q2 x
```

Entsprechend dem intuitiven Verständnis der Art und Weise, wie man eine Existenzaussage verwenden kann, schreiben wir jetzt

```
(by-assume-with "Ex-Hyp" "x" "Kernel")
```

Die Antwort ist nach einigen intern durchgeführten Zwischenschritten,

```
; ok, ?_2 can be obtained from
; ?_3: all x10.Q1 x10 & Q2 x10 -> ex x Q1 x from
;   Ex-Hyp:ex x.Q1 x & Q2 x
; ok, we now have the new goal
; ?_4: ex x Q1 x from
;   Ex-Hyp:ex x.Q1 x & Q2 x
;   x  Kernel:Q1 x & Q2 x
```

die die Anwendung der Regel der Existenz-Beseitung betreffen, schießlich die erwartete Situation

```
; ok, we now have the new goal
; ?_5: ex x Q1 x from
;   x  Kernel:Q1 x & Q2 x
```

Wir wollen jetzt die Existenz-Einführungsregel verwenden, und zwar mit x als Term:

```
(ex-intro (pt "x"))
```

Die Antwort ist

```
; ok, ?_5 can be obtained from
; ?_6: Q1 x from
;   x  Kernel:Q1 x & Q2 x
```

Es wird also ein neues Ziel $Q_1 x$ erzeugt, daß wir leicht aus der Annahme `Kernel` erhalten:

```
(use-with "Kernel" 'left)
```

Die Antwort ist

```
; ok, ?_6 is proved.  Proof finished.
```

Die Beweisausgabe mit `(cdp)` liefert hier

```
; ...(ex x10.Q1 x10 & Q2 x10) ->
     (all x10.Q1 x10 & Q2 x10 -> ex x Q1 x) ->
     ex x Q1 x by axiom Ex-Elim
; ...ex x.Q1 x & Q2 x by assumption Ex-Hyp13
; ..(all x10.Q1 x10 & Q2 x10 -> ex x Q1 x) ->
    ex x Q1 x by imp elim
; ......all x12.Q1 x12 -> ex x12 Q1 x12 by axiom Ex-Intro
; ......x
; .....Q1 x -> ex x12 Q1 x12 by all elim
; ......Q1 x & Q2 x by assumption Kernel15
; .....Q1 x by and elim left
; ....ex x12 Q1 x12 by imp elim
```

```
; ...Q1 x & Q2 x -> ex x12 Q1 x12 by imp intro Kernel15
; ..all x.Q1 x & Q2 x -> ex x12 Q1 x12 by all intro
; .ex x Q1 x by imp elim
; (ex x.Q1 x & Q2 x) -> ex x Q1 x by imp intro Ex-Hyp13
```

Bemerkung. Man kann auch im Fall der minimalen Quantorenlogik mit dem konstruktiven Existenzquantor systematisch nach einem Beweis suchen; das zugehörige Kommando ist wieder `(search)`.

KAPITEL 3

Logik für freie Algebren

Auch die Quantorenlogik ist noch nicht ausreichend, die Anfänge der Mathematik auszudrücken: wir haben weder natürliche Zahlen noch das Induktionsprinzip zur Verfügung. Wir betrachten etwas allgemeiner freie Algebren und zugehörige Prinzipien der Definition durch Rekursion und des Beweises durch Induktion. Beispiele sind die algebraischen Datenstrukturen der Listen oder Bäume. Wir formulieren ein System des natürlichen Schließens für freie Algebren (also eine Art verallgemeinerte Arithmetik), und übertragen – soweit möglich – die bisherigen Überlegungen auf diese Situation.

Im Gegensatz zur Quantorenlogik aus Kapitel 2 liegt jetzt nicht mehr ein einzelner Individuenbereich zugrunde, sondern mehrere freie Algebren, die wir jetzt *Typen* (genauer *Grundtypen*) nenne.

1. Typen und Terme

Zunächst müssen wir definieren, was (einfache) *Typen* sind. Spezielle Typen sind die sogenannten *Grundtypen*, die für freie Algebren stehen. Also

- Jeder Grundtyp μ ist ein Typ.
- Sind ρ und σ Typen, so auch $\rho \to \sigma$ (*Funktionstyp*).
- Sind ρ und σ Typen, so auch $\rho \times \sigma$ (*Paartyp*).

Beispiele. Für einen Grundtyp μ ist

$\mu \to \mu$	der Typ der einstelligen Funktionen,
$\mu \to (\mu \to \mu)$	der Typ der zweistelligen Funktionen,
$(\mu \to \mu) \to \mu$	der Typ der Funktionale.

Wir verwenden wieder naheliegende Klammerkonventionen: Außenklammern werden weggelassen, $\times$ bindet stärker als $\to$, und wir schreiben $\rho \to \sigma \to \tau$ für $\rho \to (\sigma \to \tau)$, d.h. $\to$ ist rechtsassoziativ. Man beachte, daß sich jeder nur mit $\to$ gebildete Typ ρ eindeutig schreiben läßt in der Form $\rho = \rho_1 \to \rho_2 \to \cdots \to \rho_n \to \mu$; die ρ_i nennt man die *Argumenttypen* von ρ.

Eine *freie Algebra* ist gegeben durch *Konstruktoren* beliebiger endlicher Stellenzahl. Die „Freiheit“ der Algebra besagt, daß die Konstruktoren injektiv sind und disjunkte Wertebereiche haben.

Um freie Algebren behandeln zu können, müssen wir in unserem Typsystem neben den Operatoren zur Bildung von Funktionstypen $\rho \to \sigma$ und Paartypen $\rho \times \sigma$ noch die Bildung von *induktiv erzeugten Typen* $\mu\alpha\vec{\kappa}$ zulassen. Hierbei ist α eine *Typvariable* und $\vec{\kappa} = (\kappa_i)_{i=1,\ldots,k}$ eine Liste von *Konstruktortypen*.

Zum Beispiel ist $\mu\alpha(\alpha, \alpha \to \alpha)$ der Typ der natürlichen Zahlen; hierbei steht die Liste $(\alpha, \alpha \to \alpha)$ für zwei Erzeugungsprinzipien: α für „es gibt eine

natürliche Zahl“ (die 0), und $\alpha \to \alpha$ for „für jede natürliche Zahl gibt es eine weitere“ (ihren Nachfolger).

Es sei eine abzählbar unendliche Menge von *Typvariablen* gegeben, die wir mit $\alpha, \beta, \ldots$ bezeichnen.

DEFINITION 1.1. Sei α eine Typvariable. *Typen* $\rho, \sigma, \tau, \mu \in \mathsf{T}$ und *Konstruktortypen* $\kappa \in \mathsf{KT}(\alpha)$ werden induktiv wie folgt definiert.

$$\frac{\vec{\rho} \in \mathsf{T}}{\vec{\rho} \to \alpha \to \cdots \to \alpha \to \alpha \in \mathsf{KT}(\alpha)}$$

$$\frac{\kappa_1, \ldots, \kappa_n \in \mathsf{KT}(\alpha)}{\mu\alpha\,(\kappa_1, \ldots, \kappa_n) \in \mathsf{T}} \quad (n \geq 1) \qquad \frac{\rho, \sigma \in \mathsf{T}}{\rho \to \sigma \in \mathsf{T}} \qquad \frac{\rho, \sigma \in \mathsf{T}}{\rho \times \sigma \in \mathsf{T}}$$

Hierbei steht $\vec{\rho}$ für eine Liste $\rho_1, \ldots, \rho_m$ $(m \geq 0)$ von Typen, die wir *Parametertypen* nennen. $\vec{\rho} \to \sigma$ bedeutet $\rho_1 \to \cdots \to \rho_m \to \sigma$. Wir verwenden μ ausschließlich für Typen der Form $\mu\alpha\,(\kappa_1, \ldots, \kappa_k)$. Für $\kappa[\alpha := \tau]$ schreiben wir $\kappa[\tau]$.

Wir verlangen, daß unter den Konstruktoren immer mindestens einer ist, der höchstens Parametertypen als Argumenttypen hat.

BEISPIELE. (1) Besonders einfach ist die Algebra

$$\mathbf{U} := \mu\alpha\,\alpha,$$

die nur einen einzigen Konstruktor enthält, der nullstellig ist; wir bezeichnen ihn mit ε. Die Algebra $\mathbf{U}$ besteht aus genau einem Element.

(2) Fast genauso einfach ist die Algebra

$$\mathbf{B} := \mu\alpha\,(\alpha, \alpha),$$

die genau zwei nullstellige Konstruktoren besitzt, die wir mit tt und ff bezeichnen. Die Algebra $\mathbf{B}$ besteht aus genau zwei Elementen.

(3) Das bekannteste Beispiel einer freien Algebra ist die Algebra

$$\mathbf{N} := \mu\alpha\,(\alpha, \alpha \to \alpha),$$

der *natürlichen Zahlen*; sie bildet die einfachste Algebra mit unendlich vielen Elementen. Ihre Konstruktoren sind ein nullstelliger Konstruktor 0 und ein einstelliger Konstruktor S.

(4) Wir wollen auch Algebren zulassen, die von Typparametern abhängen. Das einfachste Beispiel ist die Algebra

$$\mathbf{L}(\rho) := \mu\alpha\,(\alpha, \rho \to \alpha \to \alpha),$$

von Listen von Objekten eines gegebenen Typs ρ. Sie ist gegeben durch einen nullstelligen Konstruktor nil und einen zweistelligen Konstruktor cons. Der letztere erwartet als Argumente ein Objekt x vom Typ ρ und eine Liste l von Objekten des Typs ρ; er konstruiert daraus eine neue Liste $\mathsf{cons}(x, l)$, die durch Anhängen von x vorne an die Liste l entsteht.

(5) Eine weitere wichtige parametrisierte Algebra ist die disjunkte Vereinigung zweier durch die Typen ρ_1 und ρ_2 gegebenen Algebren. Sie wird mit $\rho_1 + \rho_2$ bezeichnet, und ist gegeben durch zwei einstellige

Konstruktoren inl und inr der Typen $\rho_1 \to \rho_1 + \rho_2$ und $\rho_2 \to \rho_1 + \rho_2$. Also

$$\rho_1 + \rho_2 := \mu\alpha\,(\rho_1 \to \alpha, \rho_2 \to \alpha)$$

(6) Schließlich kann man auch *binäre Bäume* als freie Algebra einführen durch

$$\mathbf{Bin} := \mu\alpha\,(\alpha, \alpha \to \alpha \to \alpha).$$

Alle diese Algebren sind *finitär* in dem Sinn, daß jedes Element nur endlich viele „Vorgänger“ hat. Deshalb kann man für je zwei Elemente der Algebra entscheiden, ob sie gleich sind oder nicht.

Man beachte, daß es viele gleichwertige Wege gibt, diese Typen zu definieren. Wir könnten zum Beispiel anstelle von $\mathbf{U} + \mathbf{U}$ den Typ $\mathbf{B}$ der booleschen Objekte verwenden, und anstelle von $\mathbf{L}(\mathbf{U})$ den Typ $\mathbf{N}$ der natürlichen Zahlen.

Den Paartyp $\rho \times \sigma$ haben wir nur aus Gründen der Einfachheit hinzugefügt; wir hätten ihn definieren können durch $\rho_1 \otimes \rho_2 := \mu\alpha.\rho_1 \to \rho_2 \to \alpha$.

Terme. Gegeben sei für jeden Typ ρ eine abzählbar unendliche Menge von Variablen $\Box_0^\rho, \Box_1^\rho, \ldots$ des Typs ρ; als Mitteilungszeichen verwenden wir x^ρ, y^ρ, z^ρ etc.

Definition 1.2. *Terme* werden aus getypten Variablen und den Konstanten constr_i^μ und $\mathcal{R}_\mu^\tau$ (siehe unten) induktiv definiert durch

- *Abstraktion* $(\lambda x^\rho M^\sigma)^{\rho\to\sigma}$,
- *Anwendung* $(M^{\rho\to\sigma} N^\rho)^\sigma$,
- *Paarbildung* $\langle M^\rho, N^\sigma\rangle^{\rho\times\sigma}$ und
- *Projektionen* $(M^{\rho\times\sigma}0)^\rho$, $(M^{\rho\times\sigma}1)^\sigma$.

Terme ohne Konstruktoren und Rekursionsoperatoren nennt man *reine λ-Terme*.

Wir lassen die Typindizes weg, wenn sie sich aus dem Kontaxt ergeben oder unwesentlich sind. Bei Anwendungstermen vereinbaren wir wie üblich Linksklammerung, also MNK meint $(MN)K$, nicht $M(NK)$. Ferner soll die Abstraktion stärker binden als die Anwendung, also λxMN meint $(\lambda xM)N$, nicht $\lambda x(MN)$. Wir verwenden eine Punktnotation, wenn der Wirkungsbereich eines λ-Operators so groß wie durch die Klammerung möglich sein soll, also zum Beispiel $(\lambda x.MN)K$ meint $(\lambda x(MN))K$. Wir schreiben $\lambda x_1, x_2, \ldots, x_n M$ für $\lambda x_1 \lambda x_2 \ldots \lambda x_n M$.

Für jeden Term M definieren wir wie folgt die Menge $\mathsf{FV}(M)$ aller in M *freien Variablen*.

$$\begin{aligned}
\mathsf{FV}(c) &:= \emptyset && \text{für } c = \mathsf{constr}_i^\mu \text{ oder } = \mathcal{R}_\mu^\tau \\
\mathsf{FV}(\lambda xM) &:= \mathsf{FV}(M) \setminus \{x\}. \\
\mathsf{FV}(MN) &:= \mathsf{FV}(M) \cup \mathsf{FV}(N), \\
\mathsf{FV}(\langle M, N\rangle) &:= \mathsf{FV}(M) \cup \mathsf{FV}(N), \\
\mathsf{FV}(Mi) &:= \mathsf{FV}(M) && \text{für } i \in \{0, 1\}.
\end{aligned}$$

Falls $\mathsf{FV}(M) = \emptyset$, so heißt M *geschlossener Term*.

Bemerkung. Offenbar gibt es eine enge Beziehung zwischen aussagenlogischen Herleitungen mit ihren Formeln einerseits, und Termen mit ihren

Typen andererseits. Man spricht von der CURRY-HOWARD Korrespondenz. Sie beinhaltet die Entsprechung

$$\begin{array}{lcl} \text{Formel} & \sim & \text{Typ} \\ \text{Herleitung} & \sim & \text{Term} \end{array}$$

Zum Beispiel hat man für beliebige Typen ρ, σ, τ die folgenden reinen (geschlossenen) λ-Terme:

$$\begin{aligned} \boldsymbol{i}_\rho &:= (\lambda x^\rho x^\rho)^{\rho\to\rho} \\ \boldsymbol{k}_{\rho\sigma} &:= (\lambda x^\rho \lambda y^\sigma x^\rho)^{\rho\to\sigma\to\rho} \\ \boldsymbol{s}_{\rho\sigma\tau} &:= \big(\lambda x^{\rho\to\sigma\to\tau} \lambda y^{\rho\to\sigma} \lambda z^\rho . x^{\rho\to\sigma\to\tau} z^\rho (y^{\rho\to\sigma} z^\rho)\big)^{(\rho\to\sigma\to\tau)\to(\rho\to\sigma)\to\rho\to\tau} \end{aligned}$$

Die Typen dieser Terme entsprechen offenbar den HILBERT-Axiomen

$$\begin{aligned} &A \to A, \\ &A \to B \to A, \\ &(A \to B \to C) \to (A \to B) \to A \to C, \end{aligned}$$

und die Terme entsprechen offenbar ihren (geschlossenen) Herleitungen.

Der induktiven Struktur eines Typs $\mu = \mu\alpha\,\vec{\kappa}$ entsprechen zwei Sorten von Konstanten, *Konstruktoren* und Rekursionsoperatoren. Letztere ergeben sich darus, daß Offenbar der Erzeugungsprozess der Elemente einer freien Algebra die *rekursive* Definition von Funktionen auf solchen Algebren erlaubt. Mit den Konstruktoren $\mathsf{constr}_i^\mu \colon \kappa_i[\mu]$ können wir Elemente eines Typs μ konstruieren, und mit den Rekursionsoperatoren $\mathcal{R}_\mu^\tau$ können wir Abbildungen von μ in τ definieren, und zwar durch Rekursion über die Struktur von μ. Um den Typ des Rekursionsoperators für $\mu = \mu\alpha\,\vec{\kappa}$ und den Resultat-Typ τ anzugeben, definieren wir zunächst für

$$\kappa_i = \vec{\rho} \to \alpha \to \cdots \to \alpha \to \alpha \in \mathsf{KT}(\alpha)$$

den *Schritt-Typ*

$$\begin{aligned} \delta_i^{\mu,\tau} := \vec{\rho} \to \mu \to \cdots \to \mu \to \\ \tau \to \cdots \to \tau \to \tau. \end{aligned}$$

Hierbei entsprechen $\vec{\rho}, \mu, \ldots, \mu$ den *Komponenten* des betrachteten Objekts vom Typ μ, und $\tau, \ldots, \tau$ den vorher definierten Werten. Der Rekursionsoperator $\mathcal{R}_\mu^\tau$ hat den Typ

$$\mathcal{R}_\mu^\tau \colon \delta_1^{\mu,\tau} \to \cdots \to \delta_k^{\mu,\tau} \to \mu \to \tau$$

(k ist die Anzahl der Konstruktoren).

Konversion. Um die intendierte Bedeutung des Rekursionsoperators zu verstehen, geben wir gewisse „Rechenregeln“ an, die seine Bedeutung festlegen. Es wird nützlich sein, die folgende Bezeichnungsweise zu verwenden. Sei $\mu = \mu\alpha\,\vec{\kappa}$ und

$$\kappa_i = \rho_1 \to \cdots \to \rho_m \to \alpha \to \cdots \to \alpha \to \alpha \in \mathsf{KT}(\alpha)$$

mit $n+1$ Vorkommen von α, und man betrachte $\mathsf{constr}_i^\mu \vec{N}$. Dann schreiben wir $\vec{N}^P = N_1^P, \ldots, N_m^P$ für die *Parameterargumente* $N_1^{\rho_1}, \ldots, N_m^{\rho_m}$ und $\vec{N}^R = N_1^R, \ldots, N_n^R$ für die *rekursiven Argumente* $N_{m+1}^\mu, \ldots, N_{m+n}^\mu$.

Wir definieren eine *Konversionsrelation* $\mapsto_\rho$ zwischen Termen vom Typ ρ durch

$$(\lambda x M)N \mapsto M[x := N] \tag{5}$$

$$\langle M_0, M_1\rangle i \mapsto Mi \quad (i = 0, 1) \tag{6}$$

$$\lambda x.Mx \mapsto M \quad \text{falls } x \notin \mathsf{FV}(M) \ (M \text{ keine Abstraktion}) \tag{7}$$

$$\langle M0, M1\rangle \mapsto M \quad (M \text{ kein Paar}) \tag{8}$$

$$(\mathcal{R}\vec{M})^{\mu\to\tau}(\mathsf{constr}_i^\mu \vec{N}) \mapsto M_i\vec{N}(\mathcal{R}\vec{M}N_1^R)\dots(\mathcal{R}\vec{M}N_n^R) \tag{9}$$

Die *Einschritt-Reduktionsrelation* $\mapsto$ kann man jetzt wie folgt definieren: $M \mapsto N$ soll bedeuten, daß N aus M entsteht durch Ersetzen eines Teilterms M' in M durch N', wobei $M' \mapsto N'$. Die Reduktionsrelationen $\to^+$ und $\to^*$ sind definiert als der transitive und der reflexiv-transitive Abschluß von $\mapsto$. Ein Term M ist *normal* (oder in *Normalform*), wenn es keinen Term N gibt mit $M \mapsto N$.

Man überlegt sich leicht, daß normale geschlossene Terme eines Grundtyps μ stets die Form $\mathsf{constr}_i^\mu \vec{N}$ haben.

Beispiele. (1) Für den Typ $\mathbf{B}$ der booleschen Objekte haben wir die beiden Konstruktoren $\mathsf{tt}^{\mathbf{B}} := \mathsf{constr}_1^{\mathbf{B}}$ und $\mathsf{ff}^{\mathbf{B}} := \mathsf{constr}_2^{\mathbf{B}}$, und der allgemeine Rekursionsoperator hat hier den Typ

$$\mathcal{R}_{\mathbf{B}}^\tau : \tau \to \tau \to \mathbf{B} \to \tau.$$

Wir wollen erklären, wie man mit seiner Hilfe „rekursive Definitionen“ über der Algebra $\mathbf{B}$ durchführen kann. Seien $a, b : \tau$, und $f : \mathbf{B} \to \tau$ definiert durch

$$f(\mathsf{tt}) = a \text{ und } f(\mathsf{ff}) = b.$$

Offenbar bedeutet f dasselbe wie $\mathcal{R}_{\mathbf{B}}^\tau ab$. Genauer gilt

$$\mathcal{R}_{\mathbf{B}}^\tau ab\mathsf{tt} \mapsto a \text{ und } \mathcal{R}_{\mathbf{B}}^\tau ab\mathsf{ff} \mapsto b.$$

(2) Für den Typ $\mathbf{N}$ der natürlichen Zahlen haben wir den nullstelligen Konstruktor $0^{\mathbf{N}} := \mathsf{constr}_1^{\mathbf{N}}$ und den einstelligen Konstruktor $\mathsf{S}^{\mathbf{N}\to\mathbf{N}} := \mathsf{constr}_2^{\mathbf{N}}$. Der allgemeine Rekursionsoperator hat hier den Typ

$$\mathcal{R}_{\mathbf{N}}^\tau : \tau \to (\mathbf{N} \to \tau \to \tau) \to \mathbf{N} \to \tau.$$

Wir wollen erklären, wie man mit seiner Hilfe rekursive Definitionen über der Algebra $\mathbf{N}$ durchführen kann. Seien $a : \tau$ und $h : \mathbf{N} \to \tau \to \tau$ gegeben und $f : \mathbf{N} \to \tau$ rekursiv definiert durch

$$\begin{aligned} f(0) &= a, \\ f(\mathsf{S}(n)) &= h(n, f(n)). \end{aligned}$$

Offenbar bedeutet f dann dasselbe wie $\mathcal{R}_{\mathbf{N}}^\tau ah$. Genauer gilt

$$\mathcal{R}_{\mathbf{N}}^\tau ah0 \mapsto a \text{ und } \mathcal{R}_{\mathbf{N}}^\tau ah(\mathsf{S}n) \mapsto hn(\mathcal{R}_{\mathbf{N}}^\tau ahn).$$

Zum Beispiel ist die Verdoppelungsfunktion $d : \mathbf{N} \to \mathbf{N}$ rekursiv definiert durch

$$\begin{aligned} d(0) &= 0, \\ d(\mathsf{S}n) &= \mathsf{S}(\mathsf{S}(d(n))). \end{aligned}$$

Als mit dem Rekursionsoperator gebildeter Term schreibt sie sich wie folgt:

$$\mathcal{R}^{\mathbf{N}}_{\mathbf{N}} 0(\lambda n \lambda p.\mathsf{S}(\mathsf{S}\, p)).$$

Die *Addition* $+\colon \mathbf{N} \to \mathbf{N} \to \mathbf{N}$ läßt sich rekursiv definieren durch

$$\begin{aligned} m + 0 &= m, \\ m + \mathsf{S}n &= \mathsf{S}(m+n). \end{aligned}$$

Der zugehörige Term ist

$$M_+ := \mathcal{R}^{\mathbf{N}\to\mathbf{N}}_{\mathbf{N}}(\lambda m m)\big(\lambda n \lambda p^{\mathbf{N}\to\mathbf{N}} \lambda m.\mathsf{S}(pn)\big)$$

Aus der Addition ergibt sich die *Multiplikation* $*\colon \mathbf{N} \to \mathbf{N} \to \mathbf{N}$ durch die zusätzliche rekursive Definition durch

$$\begin{aligned} m * 0 &= 0, \\ m * \mathsf{S}n &= (m * n) + m. \end{aligned}$$

Der zugehörige Term ist

$$M_* := \mathcal{R}^{\mathbf{N}\to\mathbf{N}}_{\mathbf{N}}(\lambda m 0)\big(\lambda n \lambda p^{\mathbf{N}\to\mathbf{N}} \lambda m.M_+(pm)m\big).$$

(3) Für den parametrisierten Typ $\mathbf{L}(\rho)$ der Listen von Objekten des Typs ρ haben wir den nullstelligen Konstruktor $\mathsf{nil}^{\mathbf{L}(\rho)} := \mathsf{constr}_1^{\mathbf{L}(\rho)}$ und den zweistelligen Konstruktor $\mathsf{cons}^{\rho\to\mathbf{L}(\rho)\to\mathbf{L}(\rho)} := \mathsf{constr}_2^{\mathbf{L}(\rho)}$. Der allgemeine Rekursionsoperator hat hier den Typ

$$\mathcal{R}^{\tau}_{\mathbf{L}(\rho)}\colon \tau \to (\rho \to \mathbf{L}(\rho) \to \tau \to \tau) \to \mathbf{L}(\rho) \to \tau.$$

Wir wollen wieder erklären, wie man mit seiner Hilfe rekursive Definitionen über der Algebra $\mathbf{L}(\rho)$ durchführen kann. Seien $a\colon \tau$ und $h\colon \rho \to \mathbf{L}(\rho) \to \tau \to \tau$ gegeben und $f\colon \mathbf{L}(\rho) \to \tau$ rekursiv definiert durch

$$\begin{aligned} f(\mathsf{nil}) &= a, \\ f(\mathsf{cons}(x,l)) &= h(x,l,f(l)). \end{aligned}$$

Offenbar bedeutet f dann dasselbe wie $\mathcal{R}^{\tau}_{\mathbf{L}(\rho)} ah$. Genauer gilt

$$\mathcal{R}^{\tau}_{\mathbf{L}(\rho)} ah\mathsf{nil} \mapsto a \text{ und } \mathcal{R}^{\tau}_{\mathbf{L}(\rho)} ah(\mathsf{cons}xl) \mapsto hxl(\mathcal{R}^{\tau}_{\mathbf{L}(\rho)} ahl).$$

Zum Beispiel ist die *append*-Funktion $:+:$ für Listen rekursiv definiert durch

$$\begin{aligned} \mathsf{nil} :+: l_2 &:= l_2, \\ (\mathsf{cons}\, x\, l_1) :+: l_2 &:= \mathsf{cons}\, x(l_1 :+: l_2). \end{aligned}$$

Sie läßt sich durch den folgenden Term definieren

$$M_{:+:} := \mathcal{R}^{\mathbf{L}(\alpha)\to\mathbf{L}(\alpha)}_{\mathbf{L}(\alpha)}(\lambda l_2 l_2)(\lambda x \lambda l_1 \lambda p \lambda l_2.\mathsf{cons}\, x(pl_2)).$$

Unter Verwendung der append-Funktion $:+:$ können wir das *Umdrehen* Rev von Listen definieren durch

$$\begin{aligned} \mathsf{Rev}\ \mathsf{nil} &:= \mathsf{nil}, \\ \mathsf{Rev}(\mathsf{cons}\, x\, l) &:= (\mathsf{Rev}\ l) :+: (\mathsf{cons}\, x\mathsf{nil}). \end{aligned}$$

Der entsprechende Term ist

$$M_{\mathsf{Rev}} := \mathcal{R}^{\mathbf{L}(\alpha)}_{\mathbf{L}(\alpha)}\mathsf{nil}\big(\lambda x \lambda l \lambda p. M_{:+:}\, p\,(\mathsf{cons}\, x \mathsf{nil})\big).$$

(4) Für den parametrisierten Typ $\rho_1 + \rho_2$ (disjunkte Vereinigung) haben wir zwei einstellige Konstruktoren $\mathsf{inl}_{\rho_1\rho_2} := \mathsf{constr}_1^{\rho_1+\rho_2}$ und $\mathsf{inr}_{\rho_1\rho_2} := \mathsf{constr}_2^{\rho_1+\rho_2}$ der Typen $\rho_1 \to \rho_1 + \rho_2$ und $\rho_2 \to \rho_1 + \rho_2$. Der allgemeine Rekursionsoperator hat hier den Typ

$$\mathcal{R}^{\tau}_{\rho_1+\rho_2} \colon (\rho_1 \to \tau) \to (\rho_2 \to \tau) \to \rho_1 + \rho_2 \to \tau.$$

Die Schritt-Typen $\rho_1 \to \tau$ und $\rho_2 \to \tau$ enthalten keine rekursiven Aufrufe, so daß sich das Rekursionsschema einfach schreibt als

$$f(\mathsf{inl}(x)) = g(x),$$
$$f(\mathsf{inr}(y)) = h(y).$$

Offenbar bedeutet f dann dasselbe wie $\mathcal{R}^{\tau}_{\rho_1+\rho_2} gh$. Genauer gilt

$$\mathcal{R}^{\tau}_{\rho_1+\rho_2} gh(\mathsf{inl}x) \mapsto gx \text{ und } \mathcal{R}^{\tau}_{\rho_1+\rho_2} gh(\mathsf{inr}x) \mapsto hx.$$

(5) Gesetzt wir wollen durch simultane Rekursion zwei Funktionen auf dem Typ $\mathbf{N}$ definieren, etwa $\mathsf{Even}, \mathsf{Odd} \colon \mathbf{N} \to \mathbf{B}$. Wir wollen also haben

$$\mathsf{Even}(0) := \mathsf{tt} \qquad \mathsf{Odd}(0) := \mathsf{ff}$$
$$\mathsf{Even}(\mathsf{S}\, n) := \mathsf{Odd}(n) \qquad \mathsf{Odd}(\mathsf{S}\, n) := \mathsf{Even}(n)$$

Dies läßt sich mit Hilfe von Paartypen erreichen: wir definieren rekursiv eine einzelne Funktion $\mathsf{EvenOdd} \colon \mathbf{N} \to \mathbf{B} \times \mathbf{B}$. Die Schritt-Typen sind

$$\delta_1 = \mathbf{B} \times \mathbf{B},$$
$$\delta_2 = \mathbf{N} \to \mathbf{B} \times \mathbf{B} \to \mathbf{B} \times \mathbf{B},$$

und wir können definieren $\mathsf{EvenOdd} := \mathcal{R}^{\mathbf{B}\times\mathbf{B}}_{\mathbf{N}}\langle \mathsf{tt}, \mathsf{ff}\rangle(\lambda n \lambda p. \langle p1, p0\rangle)$.

(6) Man beachte, daß zum definierten Tensorprodukt folgende Konstruktoren und folgender Rekursionsoperator gehören:

$$\big(\otimes^{+}_{\rho_1\rho_2}\big)^{\rho_1\to\rho_2\to\rho_1\otimes\rho_2} := \mathsf{constr}_1^{\rho_1\otimes\rho_2},$$
$$\mathcal{R}^{\tau}_{\rho_1\otimes\rho_2} \colon (\rho_1 \to \rho_2 \to \tau) \to \rho_1 \otimes \rho_2 \to \tau,$$

Fallunterscheidung. Eine wichtige Variante des Rekursionsschemas liegt dann vor, wenn keine rekursiven Aufrufe erfolgen. In diesem Fall haben die Schritt-Typen die Form

$$\delta_i^{\mu,\tau} := \vec{\rho} \to \mu \to \cdots \to \mu \to \tau,$$

Diese Variante heißt *Fallunterscheidungsoperator*; man unterscheidet Fälle nach der äußeren Konstruktorform. Für den Fallunterscheidungsoperator $\mathsf{Cases}^{\tau}_{\mu}$ haben wir wieder den Typ

$$\mathsf{Cases}^{\tau}_{\mu} \colon \delta_1^{\mu,\tau} \to \cdots \to \delta_k^{\mu,\tau} \to \mu \to \tau$$

(k ist die Anzahl der Konstruktoren). Die intendierte Bedeutung ergibt sich aus der Konversionsregel (vgl. (9))

$$(\mathsf{Cases}\vec{M})^{\mu\to\tau}(\mathsf{constr}_i^{\mu}\vec{N}) \mapsto M_i\vec{N}. \tag{10}$$

Das einfachste Beispiel (für den Typ $\mathbf{B}$) ist *if-then-else*. Ein anderes Beispiel ist die *Vorgängerfunktion* auf $\mathbf{N}$, also $\mathsf{P}(0) := 0$, $\mathsf{P}(\mathsf{S}(n)) := n$. Man kann sie formal definieren durch den Term

$$\mathsf{Cases}_{\mathbf{N}\to\mathbf{N}} 0 (\lambda nn).$$

BEMERKUNG. Wenn man den Wert eines Fallunterscheidungsoperators berechnet, wird man aus Effizienzgründen nicht alle Argumente auswerten wollen, sondern nur das Testargument und in Abhängigkeit von dem Resultat genau eines der anderen Argumente. Dieses Phänomen ist gut bekannt aus funktionalen Sprachen; z.B. gibt es in SCHEME ein `if` als *spezielle Form* (und nicht als Operator). Aus diesem Grund verwenden wir ein `if`-Konstrukt zum Bilden von Termen, das sich von dem Fallunterscheidungsoperator darin unterscheidet, daß die Argumente nur bei Bedarf ausgewertet werden („lazy evaluation"). Die Vorgängerfunktion läßt sich dann schreiben in der Form $\lambda m[\texttt{if}\ m\ 0\ \lambda nn]$.

Termersetzungsregeln. Die den Konstruktoren entsprechenden Eliminationskonstanten sind die Rekursionsoperatoren $\mathcal{R}$. In diesem Kontext reduziert jeder geschlossene Term eines Grundtyps auf ein Numeral.

Aus Gründen der Lesbarkeit und der Bequemlichkeit wollen wir zusätzlich Konstanten für beliebige berechenbare Funktionale verwenden, und sie entsprechend ihrer intendierten Bedeutung mit Hilfe von Termersetzungsregeln axiomatisieren. Wegen dieser Möglichkeit, eine Konstante zu „programmieren", nennen wir sie *Programmkonstante*. Offenbar kann es dann nicht mehr richtig sein, daß jeder geschlossene Term eines Grundtyps auf ein Numeral reduziert. Ferner können wir nicht-terminierende Terme erhalten.

Eine wichtige Konsequenz des Zulassens nicht-terminierender Terme ist es, daß damit unser Beweisbegriff nicht mehr entscheidbar ist: beim Prüfen, ob zwei Terme gleich sind, kann man in eine nicht-terminierende Schleife geraten. Um diese unerfreuliche Situation zu vermeiden, können wir unsere Beweise als abgekürzte Formen vollständiger Beweise auffassen, in denen gewisse Gleichheitsschlüsse implizit gelassen sind. Wenn wir dann geeignete Informationen zur Wiederherstellung des vollständigen Beweises hinzunehmen (z.B. für jeden Knoten eine Schranke für die Anzahl der Termersetzungsschritte, die für seine Verifikation erforderlich sind), erhalten wir wieder die Entscheidbarkeit.

Aber auch ohne solche Maßnahmen haben wir die Semi-Entscheidbarkeit von Beweisen, d.h. einen Algorithmus zur Beweisüberprüfung, der nur korrekte Resultate liefern kann, aber eventuell nicht terminiert. In der Praxis ist dies ausreichend.

Wir bescheiben jetzt unseren Ansatz für Termersetzungsregeln. Für jede Programmkonstante c^ρ sei eine Menge von Termersetzungsregeln der Form $c\vec{K} \mapsto N$ gegeben,wobei $\mathsf{FV}(N) \subseteq \mathsf{FV}(\vec{K})$ und $c\vec{K}$, N denselben Typ haben (nicht notwendig einen Grundtyp). Ferner verlangen wir, daß für je zwei Regeln $c\vec{K} \mapsto N$ und $c\vec{K}' \mapsto N'$ die Argumentlisten $\vec{K}$ und $\vec{K}'$ dieselbe Länge haben. Wir nennen diese Länge die *Stelligkeit* (arity) der Programmkonstanten c.

Gegeben sei eine Menge von Termersetzungsregeln. Wir wollen einige dieser Regeln – genannt *Berechnungsregeln* – in einer speziellen, besonders

effizienten Weise behandeln. Die Idee ist hier, daß eine Berechnungsregel verstanden werden kann als die Beschreibung einer Rechnung in einem „semantischen Modell“, falls die syntaktischen Konstruktoren den semantischen im Modell entsprechen, während die anderen Regeln *syntaktische* Transformationen beschreiben.

Zur Beschreibung der Berechnungsregeln benötigen wir den Begriff eines *Konstruktormusters*. Darunter verstehen wir spezielle Terme, die induktiv wie folgt definiert sind.

- Jede Variable ist ein Konstruktormuster.
- Ist c ein Konstruktor und sind $P_1, \ldots, P_n$ Konstruktormuster (oder Projektionsmarkierer 0 oder 1) so daß $c\vec{P}$ einen Grundtyp hat, so ist $c\vec{P}$ ein Konstruktormuster.

Aus der gegebenen Menge von Termersetzungsregeln wählen wir eine Teilmenge COMP mit den folgenden Eigenschaften aus:

- Falls $c\vec{P} \mapsto Q \in \text{COMP}$, so sind $P_1, \ldots, P_n$ Konstruktormuster oder Projektionsmarkierer.
- Die Regeln sind *links-linear*, d.h. falls $c\vec{P} \mapsto Q \in \text{COMP}$, so kommt jede Variable in $c\vec{P}$ nur einmal in $c\vec{P}$ vor.
- Die Regeln „überlappen“ nicht, d.h. für verschiedene Regeln $c\vec{K} \mapsto M$ und $c\vec{L} \mapsto N$ in COMP sind die linken Seiten $c\vec{K}$ und $c\vec{L}$ nicht „unifizierbar“ (in dem Sinn, daß sie durch keine Substitution gleich gemacht werden können).

Wir schreiben $c\vec{M} \mapsto_{\mathsf{comp}} Q$, falls diese Regel in COMP liegt. Alle anderen Regeln nennen wir (echte) Termersetzungsregeln, geschrieben $c\vec{M} \mapsto_{\mathsf{rew}} K$.

In unserer Reduktionsstrategie werden wir immer zuerst die Berechnungsregeln anwenden, und da sie nicht-überlappend sind, ist dieser Teil der Reduktion eindeutig. Da wir daneben fast beliebige (echte) Termersetzungsregeln zulassen, können anschließend auf einen Term verschiedene Regeln anwendbar sein; wir wählen dann die erste anwendbare Regel.

BEISPIELE. (1) Für ein durch Fallunterscheidung definiertes c haben wir die Berechnungsregeln $c\mathsf{tt} \mapsto a$ und $c\mathsf{ff} \mapsto b$.

(2) Für die Verdoppelungsfunktion $d\colon \mathbf{N} \to \mathbf{N}$ haben wir

$$\begin{aligned} d(0) &\mapsto 0, \\ d(\mathsf{S}n) &\mapsto \mathsf{S}(\mathsf{S}(d(n))), \end{aligned}$$

für die *Addition* $+\colon \mathbf{N} \to \mathbf{N} \to \mathbf{N}$

$$\begin{aligned} m + 0 &\mapsto m, \\ m + \mathsf{S}n &\mapsto \mathsf{S}(m+n) \end{aligned}$$

und für die *Multiplikation* $*\colon \mathbf{N} \to \mathbf{N} \to \mathbf{N}$

$$\begin{aligned} m * 0 &\mapsto 0, \\ m * \mathsf{S}n &\mapsto (m*n) + m. \end{aligned}$$

Entsprechend lassen sich die weiteren Definitionsgleichungen in den oben behandelten Beispielen als Berechnungsregeln auffassen.

2. Formeln

Das intendierte Modell unserer Theorie ist eine mehrsortige Struktur, mit einer Sorte für jeden Typ. Unsere Sprache ist anzusehen als eine Art Arithmetik, da wir ein spezielles intendiertes Modell beschreiben wollen.

Eine *Prädikatenvariable* (oder auch Relationsvariable) P der Stelligkeit $\rho_1, \ldots, \rho_n$ ist ein Platzhalter für eine Formel A mit ausgezeichneten (verschiedenen) Variablen $x_1, \ldots, x_n$ der Typen $\rho_1, \ldots, \rho_n$. Ein derartiges Gebilde heißt *Komprehensionsterm*, geschrieben $\{x_1, \ldots, x_n \mid A\}$. Man beachte, daß A weitere freie Variablen enthalten kann. Daneben erlauben wir *Prädikatenkonstante* mit einer festen intendierten Bedeutung. Prädikatenvariable und Konstante heißen beide *Prädikatensymbole*. Für eine mit einer Prädikatenkonstanten aufgebaute Primformel kann kein Komprehensionsterm substituiert werden.

Eine spezielle und besonders wichtige Prädikatenkonstante ist

$$\mathsf{atom}$$

der Stelligkeit $(\mathbf{B})$. Die Primformel $\mathsf{atom}(r)$ besagt, daß der Term r vom Typ $\mathbf{B}$ den Wert tt hat. Die Verwendung der Prädikatenkonstanten atom eröffnet die Möglichkeit, *entscheidbare* (d.h. rekursiv definierbare) Relationen auf der „Termebene“ ansiedeln, also als booleschwertige Funktionen auffassen. Die zugehörige atomaren Formeln sind dann von der Form $\mathsf{atom}(t)$, wobei t ein Term vom Typ $\mathbf{B}$ ist; insbesondere gilt dies für Gleichungen. Ein Vorteil dieser Sicht von atomaren Formeln ist es, daß man sich so logische Herleitungen sehr vereinfachen kann, indem man die Definitionsgleichungen der verwendeten Funktionen und Relationen – von links nach rechts gelesen – als *Termersetzungsregeln* auffasst. In unserer Logik identifizieren wir Terme, die bzgl. dieser Termersetzungsregeln die gleiche Normalform haben.

Eine Primformel hat die From $P(r_1, \ldots, r_n)$ mit einer Prädikatenvariablen oder -konstanten P und Termen $r_1, \ldots, r_n$. Wir schreiben

$$T,\ F \text{ für } \mathsf{atom}(\mathsf{tt}),\ \mathsf{atom}(\mathsf{ff}).$$

Betrachten wir als Beispiel die Behandlung der *Gleichheit* in einer (finitären) Algebra. Etwa für $\mathbf{N}$ kann man die Gleichheit $=: \mathbf{N} \to \mathbf{N} \to \mathbf{B}$ als booleschwertige Funktion rekursiv definieren durch

$$\begin{aligned}
(0 = 0) &= \mathsf{tt},\\
(0 = \mathsf{S}n) &= \mathsf{ff},\\
(\mathsf{S}n) = 0 &= \mathsf{ff},\\
(\mathsf{S}n = \mathsf{S}m) &= (n = m).
\end{aligned}$$

Diese Gleichungen fassen wir (von links nach rechts gelesen) als Termersetzungsregeln auf. Wenn man eine Gleichung $r = s$ als *Formel* benötigt, verwenden wir $\mathsf{atom}(=(r, s))$, was wir auch wieder als $r = s$ schreiben. Unsere Vereinbarung, Terme mit derselben Normalform als gleich anzusehen (zu „identifizieren“), führt dann dazu, daß wir

$$\begin{array}{lll}
0 = 0 & \text{identifizieren mit } \mathsf{atom}(\mathsf{tt}), \text{ also} & T,\\
0 = \mathsf{S}n & \text{identifizieren mit } \mathsf{atom}(\mathsf{ff}), \text{ also} & F,\\
\mathsf{S}n = 0 & \text{identifizieren mit } \mathsf{atom}(\mathsf{ff}), \text{ also} & F,
\end{array}$$

$$\mathsf{S}n = \mathsf{S}m \quad \text{identifizieren mit} \qquad\qquad n = m.$$

Es ist offensichtlich, daß man dadurch Herleitungen stark vereinfachen kann.

Formeln werden aus Primformeln aufgebaut durch

- Implikation $A \to B$,
- Konjunktion $A \wedge B$,
- Allquantifikation $\forall x^\rho A$ und
- Existenzquantifikation $\exists x^\rho A$.

Den klassischen Existenzquantor verwenden wir in einer „arithmetischen" Form: $\exists^{\mathsf{ca}} x A$ steht für $(\forall x.A \to F) \to F$.

3. Axiome

Zunächst einmal benötigen wir das *Wahrheitsaxiom* T, d.h. $\mathsf{atom}(\mathsf{tt})$.

Entsprechend dem Rekursionsschema möchte man bei logischen Überlegungen zu einer freien Algebra das *Induktionsschema* zur Verfügung haben. Das Induktionsschema für eine allgemeine freie Algebra $\mu = \mu\alpha\,\vec{\kappa}$ hat folgende Gestalt. Gegeben ist eine Zielformel $\forall x^\mu P(x)$. Für den Konstruktortyp

$$\kappa_i = \rho_1 \to \cdots \to \rho_m \to \alpha \to \cdots \to \alpha \in \mathsf{KT}(\alpha)$$

haben wir die *Schrittformel*

$$D_i := \forall y_1^{\rho_1}, \ldots, y_m^{\rho_m}, y_{m+1}^{\mu}, \ldots, y_{m+n}^{\mu}.P(y_{m+1}) \to \cdots \to P(y_{m+n}) \to P(\mathsf{constr}_i^\mu(\vec{y})).$$

Hierbei sind $\vec{y} = y_1^{\rho_1}, \ldots, y_m^{\rho_m}, y_{m+1}^{\mu}, \ldots, y_{m+n}^{\mu}$ die *Komponenten* des betrachteten Objekts $\mathsf{constr}_i^\mu(\vec{y})$ des Typs μ, und $P(y_{m+1}), \ldots, P(y_{m+n})$ die beim Beweis des Induktionsschritts zur Verfügung stehenden *Hypothesen*. Das Induktionsaxiom $\mathcal{I}_{x,A}$ mit $A = P(x)$ beweist dann die Formel

$$D_1 \to \cdots \to D_k \to \forall x^\mu\, P(x).$$

Zum Beispiel haben wir

$$\begin{aligned}
&\mathcal{I}_{p,A}\colon A[p := \mathsf{tt}] \to A[p := \mathsf{ff}] \to \forall p^{\mathbf{B}} A,\\
&\mathcal{I}_{n,A}\colon A[n := 0] \to (\forall n.A \to A[n := \mathsf{S}n]) \to \ \forall n^{\mathbf{N}} A,\\
&\mathcal{I}_{l,A}\colon A[l := \mathsf{nil}] \to (\forall x, l.A \to A[l := \mathsf{cons}(x,l)]) \to \forall l^{\mathbf{L}(\alpha)} A,\\
&\mathcal{I}_{x,A}\colon \forall y_1 A[x := \mathsf{inl}(y_1)] \to \forall y_2 A[x := \mathsf{inr}(y_2)] \to \forall x^{\rho_1+\rho_2} A.
\end{aligned}$$

Fallunterscheidung. Eine wichtige Variante des Induktionsschemas liegt dann vor, wenn keine Induktionsvoraussetzungen benutzt werden. In diesem Fall haben die Schrittformeln die Form

$$D_i := \forall y_1^{\rho_1}, \ldots, y_m^{\rho_m}, y_{m+1}^{\mu}, \ldots, y_{m+n}^{\mu}.P(\mathsf{constr}_i^\mu(\vec{y})).$$

Diese Variante heißt *Fallunterscheidungsaxiom*; es unterscheidet Fälle nach der äußeren Konstruktorform. Die Formel des Fallunterscheidungsaxioms ist

$$\mathsf{Cases}_{x,P(x)}\colon D_1 \to \cdots \to D_k \to \forall x^\mu P(x).$$

Beispiele sind

$$\begin{aligned}
&\mathsf{Cases}_{n,A}\colon A[n := 0] \to \forall n A[n := \mathsf{S}n] \to \ \forall n^{\mathbf{N}} A,\\
&\mathsf{Cases}_{l,A}\colon A[l := \mathsf{nil}] \to \forall x, l A[l := \mathsf{cons}(x,l)] \to \forall l^{\mathbf{L}(\alpha)} A.
\end{aligned}$$

4. Beispiele für Induktionsbeweise

Wir fassen die natürlichen Zahlen als eine prototypische finitäre Algebra auf. Sei n eine Variable des Typs $\mathbf{N}$ und p eine Variable des Typs $\mathbf{B}$.

Aus offensichtlichen Gründen wollen wir die *Gleichheit* als Relationssymbol zur Verfügung haben. Um damit umgehen zu können, benötigen wir die Herleitbarkeit von

$$\begin{array}{ll} n = n & \text{Reflexivität,} \\ n = m \to m = n & \text{Symmetrie,} \\ n = m \to m = k \to n = k & \text{Transitivität,} \end{array}$$

und für jedes Funktionssymbol f und Relationssymbol R muß die Herleitbarkeit der folgenden *Verträglichkeitsformeln* sichergestellt sein:

$$n_1 = m_1 \to \cdots \to n_n = m_n \to f n_1 \ldots n_n = f m_1 \ldots m_n,$$
$$n_1 = m_1 \to \cdots \to n_n = m_n \to R n_1 \ldots n_n \to R m_1 \ldots m_n.$$

Unsere Auffassung von $n = m$ als $\mathsf{atom}(=(r, s))$ mit einer rekursiv definierten booleschwertigen Funktion $=: \mathbf{N} \to \mathbf{N} \to \mathbf{B}$ zusammen mit der Identifikation von Termen mit derselben Normalform erlaubt es uns, diese Formeln durch Induktion zu beweisen.

Eine Alternative wäre es, sie als Axiome anzusehen. Dies würde jedoch dazu führen, daß vollständig angegebene Herleitungen recht lang werden.

Wir geben jetzt einige einfache Beispiele für Induktionsbeweise, und zwar von (den Allabschlüssen von)

$$\begin{aligned} & 0 + n = n \\ & n + (m + k) = n + m + k \\ & \mathsf{S}(n) + m = \mathsf{S}(n + m); \\ & n + m = m + n \end{aligned}$$

Beweis zu $\forall n\, 0 + n = n$. Induktion über n. *Basis.* $0 + 0 = 0$ reduziert zu $0 = 0$ und weiter zu T, ist also mit dem Truth-Axiom beweisbar. *Schritt.* IH: $0+n = n$. Zu zeigen ist $0+\mathsf{S}(n) = \mathsf{S}(n)$. Dies reduziert zu $\mathsf{S}(0+n) = \mathsf{S}(n)$ und weiter zu $0 + n = n$, also zu der IH. □

MPC-Beweis zu $\forall n\, 0 + n = n$. Wir verwenden eine Datei `nat0.scm`, die die Ein- und Ausgabe von Numeralen regelt (damit man z.B. 0 statt `Zero` hat).

```
MPC;
LOAD "examples/mpc/nat0.scm";

FUNCTION nat . Plus(nat nat)
{ SYNTAX + ADDOP Plus;
          n+0 -> n;
          n+Succ m -> Succ(n+m);
};

PROOF;
0+0=0;
```

```
{ n.
  { 0+n=n.
    0+Succ n=Succ n;
  }
  0+n=n -> 0+Succ n=Succ n;
}
all n.0+n=n -> 0+Succ n=Succ n;
all n 0+n=n;
```

□

BEWEIS ZU $\forall n, m, k\ n + (m + k) = n + m + k$. Wir wählen n, m beliebig aber fest. Induktion über k. *Basis.* $n + (m + 0) = n + m + 0$ reduziert zu $n + m = n + m$ und weiter zu T, ist also mit dem Truth-Axiom beweisbar. *Schritt.* IH: $n + (m + k) = n + m + k$. Zu zeigen ist $n+(m+\mathsf{S}(k)) = n+m+\mathsf{S}(k)$. Dies reduziert zu $\mathsf{S}(n+(m+k)) = \mathsf{S}(n+m+k)$ und weiter zu $n + (m + k) = n + m + k$, also zu der IH. □

MPC-BEWEIS ZU $\forall n, m, k\ n + (m + k) = n + m + k$. Wir beschränken uns auf die Angabe des eigentlichen Beweises.

```
{ n.
  { m.
    n+(m+0)=n+m+0;
    { k.
      { n+(m+k)=n+m+k.
        n+(m+Succ k)=n+m+Succ k;
      }
    }
    all k.n+(m+k)=n+m+k -> n+(m+Succ k)=n+m+Succ k;
    all k n+(m+k)=n+m+k;
  }
  all m,k n+(m+k)=n+m+k;
}
all n,m,k n+(m+k)=n+m+k;
```

□

BEWEIS ZU $\forall n, m\ \mathsf{S}(n) + m = \mathsf{S}(n + m)$. Wähle n beliebig aber fest. Induktion über m. *Basis.* $\mathsf{S}(n) + 0 = \mathsf{S}(n + 0)$ reduziert zu $\mathsf{S}(n) = \mathsf{S}(n)$ und weiter zu T, ist also mit dem Truth-Axiom beweisbar. *Schritt.* IH: $\mathsf{S}(n) + m = \mathsf{S}(n + m)$. Zu zeigen ist $\mathsf{S}(n) + \mathsf{S}(m) = \mathsf{S}(n + \mathsf{S}(m))$. Dies reduziert zu $\mathsf{S}(\mathsf{S}(n)+m) = \mathsf{S}(\mathsf{S}(n+m))$ und weiter zu $\mathsf{S}(n)+m = \mathsf{S}(n+m)$, also zu der IH. □

MPC-BEWEIS ZU $\forall n, m\ \mathsf{S}(n) + m = \mathsf{S}(n + m)$. Der eigentliche Beweis ist

```
{ n.
  Succ n+0=Succ(n+0);
  { m.
    { Succ n+m=Succ(n+m).
      Succ n+Succ m=Succ(n+Succ m);
```

```
    }
    Succ n+m=Succ(n+m) -> Succ n+Succ m=Succ(n+Succ m);
  }
  all m.Succ n+m=Succ(n+m) -> Succ n+Succ m=Succ(n+Succ m);
  all m Succ n+m=Succ(n+m);
}
all n,m Succ n+m=Succ(n+m);
```

□

Beweis zu $\forall n, m\ n + m = m + n$. Wir wählen n beliebig aber fest. Induktion über m. *Basis.* $n+0 = 0+n$ reduziert wegen der Termersetzungsregel $0 + n \mapsto n$ zu $n = n$ und weiter zu T, ist also mit dem Truth-Axiom beweisbar. *Schritt.* IH: $n + m = m + n$. Zu zeigen ist $n + \mathsf{S}(m) = \mathsf{S}(m) + n$. Dies reduziert wegen der Termersetzungsregel $\mathsf{S}(n) + m \mapsto \mathsf{S}(n + m)$ zu $\mathsf{S}(n + m) = \mathsf{S}(m + n)$ und weiter zu $n + m = m + n$, also zu der IH. □

MPC-Beweis zu $\forall n, m\ \mathsf{S}(n) + m = \mathsf{S}(n + m)$. Wir fügen die bewiesenen Gleichungen als Termersetzungsregeln der Definition von + hinzu.

```
MPC;
LOAD "examples/mpc/nat0.scm";

FUNCTION nat . Plus(nat nat)
{ SYNTAX + ADDOP Plus;
          n+0 -> n;
          n+Succ m -> Succ(n+m);
  REWRITE 0+n -> n;
  REWRITE Succ n +m -> Succ(n+m);
  REWRITE n+(m+k) -> n+m+k;   //not needed here
};

PROOF;
{ n.
  n+0=0+n;
  { m.
    { n+m=m+n.
      n+Succ m=Succ m+n;
    }
    n+m=m+n -> n+Succ m=Succ m+n;
  }
  all m.n+m=m+n -> n+Succ m=Succ m+n;
  all m n+m=m+n;
}
all n,m n+m=m+n;
```

□

Rechnergestützte Induktionsbeweise. Die Rechnerhilfe ist bei Induktionsbeweisen besonders erwünscht, da

- man auf diese Weise an die Induktionsformeln erinnert wird (die für unsere allgemeinen freien Algebren manchmal recht komplex sein können), und
- die Verwendung von Termersetzungsregeln hier besonders nützlich ist und vom Rechner gut unterstützt werden kann.

Betrachten wir zunächst die Formel

$$\forall n\, 0 + n = n,$$

deren informalen Beweis wir oben diskutiert haben.

Interaktiver Beweis. Wir beginnen die Beweiskonstruktion mit einem Aufruf des Systems sowie der Ein- und Ausgabevereinbarungen für Numerale. Es folgt die Deklaration der Addition als Programmkonstante sowie etwas Scheme-Code, der die Ein- und Ausgabe der Addition als Infix-Symbol + erlaubt.

```
> (load "~/minlog/init.scm")
(load "~/minlog/examples/mpc/nat0.scm")

(add-program-constant
 "natPlus" (py "nat=>nat=>nat") 1 'const 2)

(add-token
 "+"
 'add-op
 (lambda (x y)
   (mk-term-in-app-form
    (make-term-in-const-form
      (pconst-name-to-pconst "natPlus")) x y)))

(add-display
 (py "nat")
 (lambda (x)
   (let* ((op (term-in-app-form-to-final-op x))
          (args (term-in-app-form-to-args x)))
     (if (and (term-in-const-form? op)
              (string=? "natPlus"
                        (const-to-name
                          (term-in-const-form-to-const op)))
              (= 2 (length args)))
         (list 'add-op "+"
               (term-to-token-tree (car args))
               (term-to-token-tree (cadr args)))
         #f))))
```

Wir „animieren“ jetzt die Programmkonstante + durch Angabe der Berechnungsregeln:

```
(add-computation-rule (pt "n+0") (pt "n"))
(add-computation-rule (pt "n+Succ m") (pt "Succ(n+m)"))
```

Jetzt können wir unsere Zielformel eingeben:

```
(set-goal (pf "all n 0+n=n"))
```

Wir nehmen uns vor, den Beweis durch Induktion über n zu führen, also die außen generalisierte Variable. Das entsprechende Kommando ist

```
(ind)
```

Die Antwort besteht darin, daß die Basis und der Schritt der Induktion als neue Ziele präsentiert werden.

```
; ok, ?_1 can be obtained from
; ?_3: all n13.0+n13=n13 -> 0+Succ n13=Succ n13

; ?_2: 0+0=0
```

Das oben auf dem Stapel liegende Ziel ist zuerst zu bearbeiten. Wir verwenden das Kommando `normalize-goal`, um diese atomare Formel zu vereinfachen. Zur Erinnerung: $0 + 0 = 0$ reduziert zu $0 = 0$ und weiter zu T.

```
(normalize-goal)
```

Die Antwort ist

```
; ok, the normalized goal is
; ?_4: T
```

Dies ist mit dem Truth-Axiom beweisbar.

```
(use "Truth-Axiom")
```

Die Antwort ist

```
; ok, ?_4 is proved.  The active goal now is
; ?_3: all n13.0+n13=n13 -> 0+Succ n13=Succ n13
```

Jetzt ist also der Induktionsschritt zu bearbeiten. Wir beginnen damit, die Induktionsvariable mit `n` und die Induktionshypothese mit `IH` zu benennen.

```
(assume "n" "IH")
```

Die Antwort ist

```
; ok, we now have the new goal
; ?_5: 0+Succ n=Succ n from
;   n  IH:0+n=n
```

Wir verwenden wieder das Kommando `normalize-goal`, um die Zielformel zu vereinfachen. Zur Erinnerung: $0+\mathsf{S}(n) = \mathsf{S}(n)$ reduziert zu $\mathsf{S}(0+n) = \mathsf{S}(n)$ und weiter zu $0 + n = n$, also zu der IH..

```
(normalize-goal)
```

Die Antwort ist

```
; ok, the normalized goal is
; ?_6: 0+n=n from
;   n  IH:0+n=n
```

Jetzt ist das Ziel unmittelbar als Annahme vorhanden:

```
(use "IH")
```

Die Antwort ist

```
; ok, ?_6 is proved.  Proof finished.
```

Den erzeugten Beweis kann man sich mit dem Kommando `(cdp)` (für „check and display proof“) ausgeben lassen:

```
; ..0+0=0 -> (all n13.0+n13=n13 -> 0+Succ n13=Succ n13) ->
;            all n12 0+n12=n12 by axiom Ind
; ..T by axiom Truth-Axiom
; .(all n13.0+n13=n13 -> 0+Succ n13=Succ n13) ->
;            all n12 0+n12=n12 by imp elim
; ...0+n=n by assumption IH15
; ..0+n=n -> 0+n=n by imp intro IH15
; .all n.0+n=n -> 0+n=n by all intro
; all n12 0+n12=n12 by imp elim
```

Für die Kommutativität der Addition verwenden wir neben den (die Addition definierenden) Berechnungsregeln noch die vorher bewiesenen Gleichungen als Termersetzungsregeln:

```
(add-computation-rule (pt "n+0") (pt "n"))
(add-computation-rule (pt "n+Succ m") (pt "Succ(n+m)"))
(add-rewrite-rule (pt "0+n") (pt "n"))
(add-rewrite-rule (pt "Succ n+m") (pt "Succ(n+m)"))
```

Der interaktive Beweis besteht dann aus den Kommandos

```
(set-goal (pf "all n,m.n+m=m+n"))
(assume "n")
(ind)
(normalize-goal)
(use "Truth-Axiom")
(assume "m" "IH")
(normalize-goal)
(use "IH")
```

□

KAPITEL 4

Relationen

Eine Relation R über einer endlichen Menge kann man durch eine Matrix über dem Halbring $(\{0,1\}, \max, \cdot)$ darstellen. Der Verkettung $R \circ S$ zweier Relationen entspricht dann die Matrizenmultiplikation. Da der bekannte Algorithmus zur Multiplikation zweier $n \times n$-Matrizen die Komplexität $O(n^3)$ besitzt, ergibt sich daraus ein Algorithmus zur Berechnung der transitiven Hülle einer Relation von der Komplexität $O(n^4)$. Wir überlegen uns in diesem Kapitel, daß sich dieses Problem auch mit der Komplexität $O(n^3)$ lösen läßt. Dies leistet der Warshall-Algorithmus; er beruht auf der einfachen Idee, Wiederholungen in Pfaden zu vermeiden.

1. Allgemeine Begriffe

Sei X eine Menge. Eine *Relation* auf X ist eine Teilmenge $R \subseteq X \times X$. Die Begriffe der Reflexivität, Symmetrie, Antisymmetrie und Transitivität einer Relation sind aus den Anfängervorlesungen bekannt, ebenso die Begriffe der *transitiven Hülle*, *reflexiven Hülle* und *symmetrischen Hülle* einer Relation. Diese Begriffe kann man auch kombinieren und so etwa von der reflexiv-transitiven Hülle einer Relation sprechen.

Ferner wird der Begriff einer *Äquivalenzrelation* vorausgesetzt. Beispiel einer Äquivalenzrelation (auf der Menge $\mathcal{P}(X)$ aller Teilmengen einer gegebenen Menge X) ist der bekannte Begriff der *Gleichmächtigkeit* zweier solcher Teilmengen.

Ebenfalls als bekannt vorausgesetzt wird der Begriff der *Abzählbarkeit* einer Menge. Insbesondere gilt: Sind alle Mengen A_i für $i \in \mathbb{N}$ abzählbar, so auch ihre Vereinigung $\bigcup_{i \in \mathbb{N}} A_i$. Ein Beispiel für eine *überabzählbare* Menge ist $\mathbb{N}^{\mathbb{N}}$; dies beweist man mit dem *Cantorschen Diagonalverfahren*. Folgerung: Es gibt ein $g\colon \mathbb{N} \to \mathbb{N}$, das durch kein C-Programm berechnet werden kann.

2. Verknüpfung von Relationen und Matrizenmultiplikation

Definition 2.1. Seien R und S zwei Relationen auf der Menge X. Die *Verknüpfung* $R \circ S$ ist wie folgt definiert.

$$(x,y) \in R \circ S \iff \exists z{\in}X.(x,z) \in R \wedge (z,y) \in S.$$

Insbesondere kann man die *Potenzen* R^n einer Relation R auf X bilden:

$$\begin{aligned} R^0 &:= \Delta_X := \{\, (x,x) \mid x \in X \,\}, \\ R^1 &:= R, \\ R^2 &:= R \circ R, \\ &\vdots \end{aligned}$$

$$R^{k+1} := R^k \circ R \quad \text{für } k \geq 1.$$

LEMMA 2.2. *Sei R eine Relation auf X. Dann gilt für die transitive Hülle R^+ von R*

$$R^+ = \bigcup_{n=1}^{\infty} R^n.$$

Für die reflexiv-transitive Hülle R^* von R gilt

$$R^* = \bigcup_{n=0}^{\infty} R^n.$$

BEWEIS. Übung. □

Ist R reflexiv (d.h. gilt $\Delta_X \subseteq R$), so hat man $R \subseteq R^2 \subseteq R^3 \ldots$.

DEFINITION 2.3. (Matrix einer Relation). Sei R eine Relation auf der Menge $X = \{x_1, x_2, \ldots, x_n\}$, x_i paarweise verschieden. Dann kann R durch folgende $n \times n$-Matrix $M_R = (r_{ij})$ beschrieben werden:

$$r_{ij} := \begin{cases} 1 & \text{falls } (x_i, x_j) \in R \\ 0 & \text{sonst.} \end{cases}$$

BEISPIEL 2.4. Sei

$$\begin{aligned} X &:= \{x_1, x_2, x_3, x_4\}, \\ R &:= \{(x_1, x_2), (x_2, x_3), (x_3, x_4), (x_4, x_1), (x_2, x_4)\}. \end{aligned}$$

Dann ist

$$M_R = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Wir wollen jetzt die Verknüpfung zweier Relationen mit der Matrizenmultiplikation über Halbringen in Verbindung bringen. Zur Erinnerung schreiben wir noch einmal die Definition eines kommutativen Ringes auf.

DEFINITION 2.5. Ein kommutativer *Ring* ist ein Tripel $(A, +, \cdot)$ bestehend aus einer Menge A mit zwei Verknüpfungen

$$\begin{aligned} +\colon\ & A \times A \to A \\ & (x, y) \mapsto x + y \end{aligned} \quad \text{und} \quad \begin{aligned} \cdot\colon\ & A \times A \to A \\ & (x, y) \mapsto x \cdot y \end{aligned}$$

so daß folgende Axiome erfüllt sind.

I: $(A, +)$ ist eine abelsche Gruppe mit einem neutralen Element $0 \in A$. Insbesondere gelten das Assoziativgesetz, das Kommutativgesetz und man hat zu jedem Element ein Negatives.

II: $(A, \cdot)$ ist eine abelsche Halbgruppe, d.h. es gelten das Assoziativgesetz und das Kommutativgesetz für die Multiplikation.

III: Es gilt das Distributivgesetz $(x + y) \cdot z = x \cdot z + y \cdot z$ für alle $x, y, z \in A$.

Ein Element $1 \in A$ heißt *Einselement* von A, wenn gilt $1 \cdot x = x$ für alle $x \in A$.

Beispiele sind der Ring $\mathbb{Z}$ der ganzen Zahlen und jeder Körper, z.B. $\mathbb{Q}$.

DEFINITION 2.6. Ein kommutativer *Halbring* ist ein Tripel $(A, \vee, \wedge)$ bestehend aus einer Menge A mit zwei Verknüpfungen

$$\begin{aligned} \vee\colon\ & A \times A \to A \\ & (x,y) \mapsto x \vee y \end{aligned} \quad \text{und} \quad \begin{aligned} \wedge\colon\ & A \times A \to A \\ & (x,y) \mapsto x \wedge y \end{aligned}$$

so daß folgende Axiome erfüllt sind.

I: $(A, \vee)$ ist eine abelsche Halbgruppe mit einem neutralen Element $0 \in A$, d.h. es gilt

a.	$(x \vee y) \vee z = x \vee (y \vee z)$	für alle $x, y, z \in A$,
b.	$x \vee y = y \vee x$	für alle $x, y \in A$,
c.	$x \vee 0 = 0$	für alle $x \in A$.

II: $(A, \wedge)$ ist eine abelsche Halbgruppe, d.h. es gilt

a.	$(x \wedge y) \wedge z = x \wedge (y \wedge z)$	für alle $x, y, z \in A$,
b.	$x \wedge y = y \wedge x$	für alle $x, y \in A$.

III: Es gilt das Distributivgesetz $(x \vee y) \wedge z = (x \wedge z) \vee (y \wedge z)$ für alle $x, y, z \in A$. Ferner gilt $x \wedge 0 = 0$ für alle $x \in A$.

Ein Element $1 \in A$ heißt *Einselement* von A, wenn gilt $1 \wedge x = x$ für alle $x \in A$.

BEISPIEL 2.7. Sei X eine beliebige Menge und $A = \mathcal{P}(X)$ die Menge aller Teilmengen von X (also die *Potenzmenge* von X). Dann ist $(A, \cup, \cap)$ ein kommutativer Halbring. Neutrales Element ist die leere Menge $\emptyset$, und Einselement ist die Gesamtmenge X.

BEISPIEL 2.8. $A = \{0, 1\}$ und

$x \vee y := \max(x, y)$,

$x \wedge y := \min(x, y) \quad (= xy$ mit der gewöhnlichen Multiplikation in $\mathbb{N})$.

Wir können 0 und 1 als falsch und wahr verstehen, und dann $\wedge$ und $\vee$ als logisches „und“ und „oder“. Ebenso ist eine mengentheoretische Interpretation möglich: Für $X := \{*\}$ betrachte man $\mathcal{P}(X) = \{\emptyset, \{*\}\}$. Dann entspricht die leere Menge $\emptyset$ der 0 und die Einermenge $\{*\}$ der 1.

DEFINITION 2.9. (Matrizenmultiplikation über Halbringen). Sei $(A, \vee, \cdot)$ ein kommutativer Halbring mit Einselement 1. Seien $M_1 = (a_{ij})_{1 \le i,j \le n}$ und $M_2 = (b_{ij})_{1 \le i,j \le n}$ zwei $n \times n$-Matrizen mit Elementen $a_{ij}, b_{ij} \in A$. Dann wird das *Produkt*

$$M := M_1 M_2$$

definiert als

$$M = (c_{ij})_{1 \le i,j \le n} \quad \text{mit} \quad c_{ij} = \bigvee_{k=1}^{n} a_{ik} b_{kj}.$$

Dabei ist $\bigvee_{k=1}^{n} x_k$ eine Abkürzung für $x_1 \vee \cdots \vee x_k$.

Sei $(A, \vee, \cdot)$ ein kommutativer Halbring mit Einselement 1. Dann ist das Matrizenprodukt in $M(n \times n, A)$ offenbar assoziativ; neutrales Element ist

$$E_n := \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix}$$

SATZ 2.10. *Seien R und S zwei Relationen auf einer endlichen Menge $X = \{x_1, \ldots, x_n\}$ und M_R, M_S die zugehörigen Matrizen aus $M(n \times n, A)$ mit $A = \{0, 1\}$. Dann gilt für die Matrix $M_{R \circ S}$ der Verknüpfung von R mit S*

$$M_{R \circ S} = M_R M_S,$$

wobei die Matrizenmultiplikation über dem Halbring $(\{0,1\}, \vee, \cdot)$ mit $x \vee y :=$ $\max(x, y)$ zu verstehen ist.

BEWEIS. Sei $M_R = (a_{ij})$ und $M_S = (b_{ij})$ mit

$$a_{ij} = \begin{cases} 1 & \text{falls } (x_i, x_j) \in R, \\ 0 & \text{sonst,} \end{cases}$$

und analog für b_{ij}. Ferner sei $M_{R \circ S} = (c_{ij})$. Dann gilt

$$\begin{aligned} c_{ij} = 1 &\iff (x_i, x_j) \in R \circ S \\ &\iff \exists k.(x_i, x_k) \in R \wedge (x_k, x_j) \in S \\ &\iff \exists k.a_{ik} = 1 \wedge b_{kj} = 1. \end{aligned}$$

Sei andererseits $M_R M_S = (c'_{ij})$. Dann gilt

$$\begin{aligned} \left(c'_{ij} = \bigvee_k a_{ik} b_{kj} = \max_k (a_{ik} b_{kj}) = 1\right) &\iff \exists k \; a_{ik} b_{kj} = 1 \\ &\iff \exists k.a_{ik} = 1 \wedge b_{kj} = 1. \end{aligned}$$

Also ist $c_{ij} = c'_{ij}$. □

Wir beschreiben jetzt einen Algorithmus zur Multiplikation von Matrizen über dem Halbring $(\{0,1\}, \vee, \cdot)$. Sei $M = (a_{ij})_{1 \le i,j \le n}$, also

$$M = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix}$$

Die Zeilen $a_i = (a_{i1}, \ldots, a_{in})$ werden als Bitvektoren aufgefaßt. Für zwei Bitvektoren $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ und $y = (y_1, \ldots, y_n) \in \{0,1\}^n$ bezeichne $x \vee y \in \{0,1\}^n$ den Bitvektor

$$x \vee y := (x_1 \vee y_1, \ldots, x_n \vee y_n).$$

Gegeben seien also zwei Matrizen $M_1, M_2 \in M(n \times n, \{0,1\})$ mit den Zeilen

$$M_1 = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \quad \text{und} \quad M_2 = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}.$$

Gesucht ist

$$M_1 \cdot M_2 = M_3 = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}.$$

Die c_i können durch den folgenden Algorithmus berechnet werden.

```
for i := 1 to n do
   c_i := 0
   for j := 1 to n do
      if a_ij = 1 then
         c_i := c_i ∨ b_j
      end;
   end;
end;
```

Wir wollen uns jetzt die Korrektheit dieses Algorithmus überlegen. In den n Durchläufen der äußeren Schleife werden die Zeilen c_i unabhängig voneinander berechnet. Es genügt also, nur einen Durchlauf für ein festes i zu betrachten. Sei $c_i^{(j)}$ der Inhalt des Speicherplatzes für c_i nach dem j-ten Durchlauf der inneren Schleife. Vor dem ersten Durchlauf der inneren Schleife (mit Laufvariable j) hat man $c_i^{(0)} = 0$. Für $j > 0$ gilt

$$c_{ik}^{(j)} = c_{ik}^{(j-1)} \vee a_{ij} b_{jk} \quad \text{für } k = 1 \dots n.$$

Daraus folgt durch Induktion $c_{ik}^{(n)} = \bigvee_{j=1}^{n} a_{ij} b_{jk}$. Dies ist aber die (i,k)-te Komponente des gesuchten Matrizenprodukts.

Die Komplexität des Algorithmus ist $O(n^3)$. (Dies gilt, falls jede Bitoperation einzeln gezählt wird. Hat man die Möglichkeit, die Bitoperationen auf einem Bitvektor parallel durchzuführen, so ist die Komplexität $O(n^2)$). Man beachte, daß die Komplexität der gewöhnlichen Matrizenmultiplikation für $n \times n$-Matrizen ebenfalls $O(n^3)$ ist.

Wir wollen nun die Komplexität eines naiven Algorithmus zur Berechnung der transitiven Hülle abschätzen.

LEMMA 2.11. *Sei $X = \{x_1, \dots, x_n\}$ eine endliche Menge und R eine Relation auf X. Dann gilt*

$$R^+ = \bigcup_{k=1}^{n} R^k.$$

BEWEIS. Übung. □

Sei M_R die zu R gehörige Matrix. Dann ist offenbar

$$M_{R^+} = \bigvee_{k=1}^{n} M_R^k.$$

Dabei wird $\vee$ (= max) komponentenweise auf die einzelnen Matrizen angewandt. Aufgrund der obigen Abschätzung des Matrizenmultiplikations-Algorithmus kommt man also auf die Komplexität $O(n^4)$.

3. Transitive Hülle: der Warshall-Algorithmus

Der Warshall-Algorithmus berechnet die transitive Hülle R^+ einer Relation R. Während der naive Algorithmus eine Komplexität $O(n^4)$ hat, kommt der Warshall-Algorithmus mit der Komplexität $O(n^3)$ aus.

Wir beschreiben zunächst die Idee des Warshall-Algorithmus. Gegeben ist eine Relation R auf einer n-elementigen Menge $X = \{x_1, \dots, x_n\}$. Zu

vorgelegtem x_j, x_k wollen wir entscheiden, ob es einen *R-Pfad* von x_j nach x_k gibt, d.h. ob es $z_1, \ldots, z_m \in X$ gibt mit $x_j R z_1 R z_2 R \ldots z_{m-1} R z_m R x_k$. Die Grundidee des Warshall-Algorithmus ist nun folgende:

Vermeide Wiederholungen bei den Zwischenpunkten $z_1, \ldots, z_m$.

Wir definieren zunächst Relationen $R^{(i)}$ so daß $(x_j, x_k) \in R^{(i)}$ genau dann, wenn es einen R-Pfad von x_j nach x_k gibt, der nur Zwischenpunkte aus $\{x_1, \ldots, x_i\}$ verwendet. Sei also $R^{(0)} := R$ und

$$(x_j, x_k) \in R^{(i)} \quad :\Longleftrightarrow \quad (x_j, x_k) \in R^{(i-1)} \quad \text{oder} \quad x_j R^{(i-1)} x_i R^{(i-1)} x_k.$$

Offenbar gilt dann wie gewünscht

$$\begin{aligned} &(x, y) \in R^{(i)} \\ &\Longleftrightarrow\ (x, y) \in R \text{ oder } \exists z_1, \ldots, z_m \in \{x_1, \ldots, x_i\}\ x R z_1 R z_2 \ldots z_{m-1} R z_m R y, \end{aligned}$$

also auch $R^{(n)} = R^+$.

Der Warshall-Algorithmus konstruiert Matrizen $A^{(i)}$, die die Relationen $R^{(i)}$ darstellen. Nach n Schritten hat man dann also die gesuchte Matrix $A^{(n)}$ der Relation $R^{(n)} = R^+$. Sei $A^{(0)}$ die darstellende Matrix A der gegebenen Relation R, d.h. $A \in M(n \times n, \{0, 1\})$ mit den Zeilen $a_1, \ldots, a_n$:

$$A = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}.$$

Die Zeilen sind Bitvektoren; sie werden durch den Algorithmus verändert.

```
for i := 1 to n do
   for j := 1 to n do
      if a_ji = 1 then
         a_j := a_j ∨ a_i
      end;
   end;
end;
```

Hier sind in $a_j := a_j \vee a_i$ mit a_j, a_i auf der rechten Seite die Bitvektoren des *vorigen* Durchlaufs (also für $i - 1$) der äußeren Schleife gemeint.

Sei $A^{(i)}$ der Inhalt des Speicherplatzes für A nach dem i-ten Durchlauf der äußeren Schleife. Wir zeigen $A^{(i)} = M_{R^{(i)}}$ durch Induktion über i. Nach Annahme ist $A = A^{(0)} = M_{R^{(0)}}$. Aufgrund des Algorithmus haben wir

$$a_{jk}^{(i)} = a_{jk}^{(i-1)} \vee a_{ji}^{(i-1)} a_{ik}^{(i-1)}$$

für alle j, i, k. Mit Hilfe der IH $A^{(i-1)} = M_{R^{(i-1)}}$ erhält man

$$\begin{aligned} a_{jk}^{(i)} = 1 \quad &\Longleftrightarrow \quad a_{jk}^{(i-1)} = 1 \quad \text{oder} \quad \big(a_{ji}^{(i-1)} = 1 \quad \text{und} \quad a_{ik}^{(i-1)} = 1\big) \\ &\Longleftrightarrow \quad (x_j, x_k) \in R^{(i-1)} \quad \text{oder} \quad x_j R^{(i-1)} x_i R^{(i-1)} x_k \quad \text{nach IH} \\ &\Longleftrightarrow \quad (x_j, x_k) \in R^{(i)} \quad \text{nach Definition der } R^{(i)}, \end{aligned}$$

also $A^{(i)} = M_{R^{(i)}}$. Wegen $R^{(n)} = R^+$ ist $A^{(n)} = M_{R^{(n)}}$ die darstellende Matrix der transitiven Hülle von R. Damit haben wir die Korrektheit des Warshall-Algorithmus gezeigt; seine Komplexität ist offenbar $O(n^3)$ (bzw.

$O(n^2)$, wenn man die Möglichkeit hat, die Bitoperationen auf einem Bitvektor parallel durchzuführen).

4. Ein dem Warshall-Algorithmus entsprechender Beweis

Ein (konstruktiver) Beweis einer Existenzaussage enthält immer ein Verfahren, das als existent nachgewiesene Objekt zu konstruieren. Aus einem formalen solchen Beweis kann man ein entsprechendes Programm automatisch extrahieren. Ein Vorteil dieser Methode zur Erzeugung von Programmen ist es, daß man Beweise maschinell auf ihre Korrektheit überprüfen kann, was bei Programmen prinzipiell nicht möglich ist. Diesen Ansatz werden wir gleich genauer ausführen. Vorbereitend dazu soll zunächst auf der informellen Ebene gezeigt werden, wie man den Warshall-Algorithmus aus dem Beweis einer geeigneten Existenzaussage gewinnen kann. Um den „raffinierten“ Algorithmus (der ja im Gegensatz zu dem „naiven“ Algorithmus die Komplexität $O(n^3)$ anstelle von $O(n^4)$ hat) aus einem Beweis gewinnen zu können, muß der Beweis die Idee widerspiegeln, nur wiederholungsfreie Pfade zu betrachten.

Zu beachten ist, daß es sich bei unserem Beweis nicht mehr um einen solchen in der „reinen Logik“ handelt wird, da wir mit freien Algebren, d.h. induktiv erzeugten Datentypen arbeiten (den natürlichen Zahlen und Listen natürlicher Zahlen) und Beweise durch Induktion über diese Datentypen führen müssen. Wir stützen uns hier auf die in Kapitel 3 bereit gestellten Hilfsmittel.

Folgende Schritte sind auszuführen.

(a) Festlegung der verwendeten Datenstrukturen.
(b) Wahl einer Sprache mit entscheidbaren Prädikaten- und Funktionssymbolen.
(c) Auflistung der verwendeten Eigenschaften („Lemmata“) ohne „rechnerischen Gehalt“; meist sind dies rein generalisierte Allaussagen, und
(d) Führen eines konstruktiven Beweises des betrachteten $\forall\exists$-Satzes.

Diese Schritte wollen wir jetzt in unserem Beispiel durchführen.

(a) Festlegung der verwendeten Datenstrukturen. Außer dem Typ $\boldsymbol{N}$ der natürlichen Zahlen verwenden wir den Typ $\boldsymbol{L}(\boldsymbol{N})$ der (endlichen) Listen natürlicher Zahlen; sie repräsentieren Pfade mit natürlichen Zahlen als Elementen. Natürliche Zahlen bezeichenen wir mit a, b, c, i, j, k. Listen x, y, z werden erzeugt aus der leeren Liste ϵ mittels des Konstruktors cons, der aus a und x die Liste $\mathsf{cons}(a, x)$ erzeugt, die durch Anhängen von a vorne an x entsteht. Wir schreiben cons als Infixoperator und verwenden die Bezeichnung $a :: x$. Für $a :: \epsilon$ schreiben wir $a{:}$.

Wie üblich soll der Infixoperator $\cdot :: \cdot$ rechtsassoziativ sein; wir schreiben also $a :: b :: x$ für $a :: (b :: x)$.

(b) Unsere Sprache besteht aus den folgenden Prädikaten- und Funktionssymbolen. Zugrunde liegt eine zweistellige Relation R auf $\{0, 1, \ldots, n-1\}$, deren transitive Hülle zu bestimmen ist.

$k \in x$	k kommt im Pfad x vor,
$\mathsf{Rf}(x)$	x ist ein wiederholungsfreier Pfad,
$P_i(x, j, k)$	x ist ein R-Pfad von j nach k mit inneren Elementen $< i$,

$x \mid y$ Verkettung.

Die Verkettungsfunktion $x \mid y$ leistet folgendes. Sind x und y Pfade mit gleichem End- bzw. Anfangspunkt, so ist $x \mid y$ der durch Aneinanderhängen entstehende Pfad, wobei der gleiche End- und Anfangspunkt nur einmal aufgeführt wird. Sind End- und Anfangspunkt verschieden, so ist das Ergebnis die leere Liste ϵ.

Die Definitionen dieser Prädikaten- und Funktionssymbole geben wir in MPC-Syntax an, und zwar in der Form von Berechnungsregeln:

```
FUNCTION boole . In(nat (list nat))
  // k in x: k occurs in the path x
{ SYNTAX in RELOP In;
  a in nil -> False;
  a in (b::x) -> [if (a=b) True (a in x)];
};

FUNCTION boole . Rf(list nat)
  // Rf x : x is a repetition free path
{ Rf nil -> True;
  Rf(a::x) -> [if (a in x) False (Rf x)];
};

FUNCTION boole . Path(nat=>nat=>boole nat (list nat) nat nat)
{  SYNTAX P CONST (Path r);
  // we use P and suppress the notation of r
   P i nil b c -> False;
   P i (a::nil) b c -> [if (a=b) (b=c) False];
   P i (a::d::nil) b c ->
         [if (a=b) [if (d=c) (r a d) False]  False];
   P i (a::d::e::x) b c ->
         [if (a=b)
             [if (r a d)
                 [if (d<i) (P i (d::e::x) d c) False]
                 False]
             False];
 };

FUNCTION list nat . Dot((list nat) (list nat))
{SYNTAX | PAIROP Dot;
 nil | x -> x;
 (a::nil) | nil -> a:;
 (a::nil) | (b::x) -> [if (a=b) (b::x) nil];
 (a::d::x) | y -> a::(d::x | y);
};
```

(c) Auflistung der verwendeten Eigenschaften („Lemmata“) ohne „rechnerischen Gehalt“. Dies sind in unserem Fall die folgenden Aussagen; wir schreiben sie wie üblich ohne die äußeren Allquantoren auf.

(11) $P_i(x, j, k) \to P_{i+1}(x, j, k)$

(12) $P_0(x,j,k) \to j \neq k \to R(j,k)$

(13) $P_i(x,j,i) \to P_i(y,i,k) \to P_{i+1}(x \mid y,j,k)$

(14) $P_i(x,j,i) \to \mathsf{Rf}(x) \to P_i(y,i,k) \to \mathsf{Rf}(y) \to \forall z \neg P_i(z,j,k) \to \mathsf{Rf}(x \mid y)$

(15) $P_{i+1}(x,j,k) \to \neg P_i(x,j,k) \to \exists^{\mathsf{cl}} y\ P_i(y,j,i)$

(16) $P_{i+1}(x,j,k) \to \neg P_i(x,j,k) \to \exists^{\mathsf{cl}} z\ P_i(z,i,k)$

Die letzten beiden Aussagen enthalten die Hauptidee des Beweises: wenn es einen Pfad von j nach k gibt mit inneren Elementen $< i+1$, aber nicht $< i$, dann muss es Pfade von j nach i und von i nach k geben (klassischer Existenzquantor!) deren innere Elemente $< i$ sind. Die Aussage (14) kann man wie folgt einsehen: unter den gegebenen Annahmen kann $x \mid y$ nur dann eine Wiederholung enthalten, wenn x und y ein gemeinsames inneres Element haben. Dies widerspricht aber der Annahme $\forall z \neg P_i(z,j,k)$.

(d) Führen eines konstruktiven Beweises des betrachteten $\forall\exists$-Satzes. Wir formulieren zunächst die Behauptung, daß für unsere Relation R (die wir in den Bezeichnungen nicht explizit mitführen) und gegebene i,j,k einen Pfad x mit folgenden Eigenschaften gibt.

- Wenn x die leere Liste ϵ ist, so kann es keinen R-Pfad von j nach k mit inneren Elementen $< i$ geben, und
- wenn x nicht die leere Liste ϵ ist, so verbindet x die Knoten j und k, hat innere Elemente $< i$ und ist wiederholungsfrei.

PROPOSITION 4.1 (Warshall).

$$\forall i,j,k \exists x. \big(x = \epsilon \to \forall y \neg P_i(y,j,k)\big) \wedge \big(x \neq \epsilon \to P_i(x,j,k) \wedge \mathsf{Rf}(x)\big).$$

BEWEIS. Induktion über i. *Basis* $i = 0$. Gegeben j,k.

Fall $j = k$. Wähle $x := j{:}$. Dann gilt $P_0(j{:},j,k)$ wegen $j = k$ und $\mathsf{Rf}(j{:})$ nach Definition.

Fall $j \neq k$.

UFall $R(j,k)$. Wähle $x := j :: k{:}$. Dann gilt $P_0(j :: k{:},j,k)$ wegen $R(j,k)$ und $\mathsf{Rf}(j :: k{:})$ wegen $j \neq k$.

UFall $\neg R(j,k)$. Wähle $x := \epsilon$. Zu zeigen ist $\forall y \neg P_0(y,j,k)$. Sei y beliebig. Annahme: $P_0(y,j,k)$. Dann gilt $R(j,k)$ nach (12); Widerspruch.

Schritt $i+1$. Gegeben j,k. Nach IH haben wir x_i. Gesucht: x_{i+1}.

Fall $x_i = \epsilon$. Dann gilt $\forall y \neg P_i(y,j,k)$. Nach IH haben wir $x_{i,j,i}$ und $x_{i,i,k}$.

UFall $x_{i,j,i} = \epsilon$. Dann gilt $\forall y \neg P_i(y,j,i)$. Setze $x_{i+1} = \epsilon$. Gegeben y. Annahme: $P_{i+1}(y,j,k)$. Nach (15) (wegen $\neg P_i(y,j,k)$) gilt $\exists^{\mathsf{cl}} y\ P_i(y,j,i)$. Widerspruch.

UFall $x_{i,j,i} \neq \epsilon$.

UUFall $x_{i,i,k} = \epsilon$. Dann gilt $\forall y \neg P_i(y,i,k)$. Setze $x_{i+1} = \epsilon$. Gegeben y. Annahme: $P_{i+1}(y,j,k)$. Nach (16) (wegen $\neg P_i(y,j,k)$) gilt $\exists^{\mathsf{cl}} z\ P_i(z,i,k)$. Widerspruch.

UUFall $x_{i,i,k} \neq \epsilon$. Setze $x_{i+1} := x_{i,j,i} \mid x_{i,i,k}$. Dann gilt $P_{i+1}(x_{i+1},j,k)$ nach (13) und $\mathsf{Rf}(x_{i+1})$ nach (14), denn $\forall y \neg P_i(y,j,k)$ folgt aus $x_i = \epsilon$.

Fall $x_i \neq \epsilon$. Wähle $x_{i+1} := x_i$. Dann gilt $\mathsf{Rf}(x_{i+1})$ und $P_i(x_{i+1},j,k)$, also $P_{i+1}(x_{i+1},j,k)$ nach (11). $\square$

Aus einem formalen Beweis dieses Satzes kann man das folgende Programm extrahieren. Es handelt sich um die Orignalausgabe von MINLOG (bis auf Zeilenumbrüche, Variablennamen und `eps` anstelle von `(Nil nat)`):

```
[r](Rec nat=>nat=>nat=>list nat)
([j,k][if (j=k) (j:) [if (r j k) (j::k:) eps]])
([i,f,j,k][if (f j k=eps)
              [if (f j i=eps)
                  eps
                  [if (f i k=eps) eps (f j i|f i k)]]
              (f j k)])
```

Man beachte, daß der Algorithmus in genau der angegebenen Form exponentiell ist, da er im Schrittfall die drei rekursiven Aufrufe $f_i(j,k)$, $f_i(j,i)$ und $f_i(i,k)$ enthält. Mit einer bekannten Optimierungstechnik erhält man aber einen polynomialen Algorithmus: man fasse f_i auf als eine $n \times n$-Matrix (also *nicht* als eine Funktion).

KAPITEL 5

Graphen

Einen ungerichteten Graphen G über einer endlichen Menge kann man durch seine Adjazenzmatrix über dem Halbring $(\{0,1\}, \max, \cdot)$ darstellen. Aus dem Warshall-Algorithmus ergibt sich demnach ein Algorithmus der Komplexität $O(n^3)$ zur Feststellung des Zusammenhangs eines Graphen.

Mit Hilfe des Begriffs des Grades eines Knotens geben wir ein hinreichendes und notwendiges Kriterium dafür an, daß Eulersche Wege und Zyklen in einem Graphen existieren. Aus der Notwendigkeit des Kriteriums folgt, daß das Königsberger Brückenproblem unlösbar ist.

Weiter behandeln wir Abstände in bewerteten Graphen. Der einfachste Abstandsbegriff ist die Länge des kürzesten Pfades zwischen zwei Knoten. Mit dem linearen Algorithmus von Moore bestimmen wir zu einem festen Knoten v_0 und einem gegebenen Abstand k alle Knoten, die in diesem Sinn von v_0 den Abstand k haben, also die von v_0 aus durch einen Pfad der Länge höchstens k erreicht werden können.

Ein feinerer Abstandsbegriff stützt sich auf eine vorgegebene Bewertungsfunktion, die jeder Kante eines Graphen eine feste rationale Zahl (oder ∞) als Kantenlänge zuordnet. Mit einer einfachen Verallgemeinerung des Warshall-Algorithmus kann man dann zu je zwei Knoten ihren Abstand berechnen, d.h. die Länge der kürzesten (im Sinne der aufsummierten Kantenlängen) Verbindung zwischen ihnen; die Komplexität ist wieder $O(n^3)$. Abschließend behandeln wir den Dijkstra-Algorithmus zur Berechnung der Abstände von einem festen Knoten v_0; seine Komplexität ist $O(n^2)$.

1. Allgemeine Begriffe

Definition 1.1. Ein (ungerichteter) *Graph* ist ein Paar $\Gamma = (V, E)$ bestehend aus einer Menge V von *Knoten* oder *Ecken* (englisch *vertex*) und einer Menge $E \subseteq \mathcal{P}(V)$ von zweielementigen Teilmengen von V. Die Elemente von E heißen *Kanten* (englisch *edge*). Ist $e = \{A_0, A_1\}$, so sagt man, daß die Kante e die Knoten A_0 und A_1 verbindet.

Ein Beispiel eines Graphen ist in Abbildung 1 skizziert.

Man kann endliche Graphen auf verschiedene Weise durch Matrizen darstellen.

Definition 1.2. (*Adjazenzmatrix* eines endlichen Graphen). Sei $\Gamma = (V, E)$ ein endlicher Graph mit $V = \{A_1, \ldots, A_n\}$. Dann kann Γ durch folgende Matrix $M_\Gamma = (a_{ij})_{1 \le i,j \le n} \in M(n \times n, \{0,1\})$ beschrieben werden:

$$a_{ij} := \begin{cases} 1 & \text{falls } \{A_i, A_j\} \in E, \\ 0 & \text{sonst.} \end{cases}$$

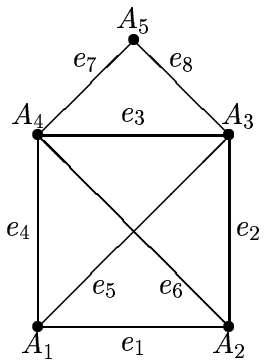


ABBILDUNG 1. Beispiel eines Graphen

Die Adjazenzmatrix eines (ungerichteten) Graphen ist offenbar symmetrisch. Ferner sind alle Diagonalelemente $a_{ii} = 0$. Im Beispiel des Graphen in Abbildung 1 ist die Adjazenzmatrix

$$M_\Gamma = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Man beachte, daß die Adjazenzmatrix eines Graphen $\Gamma = (V, E)$ auch die Matrix einer gewissen symmetrischen, antireflexiven Relation auf V ist, nämlich der Relation R_Γ definiert durch

$$R_\Gamma := \{\, (A, B) \mid \{A, B\} \in E \,\}.$$

DEFINITION 1.3. (*Inzidenzmatrix* eines endlichen Graphen). Sei $\Gamma = (V, E)$ ein endlicher Graph mit $V = \{A_1, \ldots, A_n\}$, $E = \{e_1, \ldots, e_m\}$. Dann kann Γ durch folgende Matrix $\mathrm{IM}_\Gamma = (c_{ij}) \in M(n \times m, \{0, 1\})$ beschrieben werden:

$$c_{ij} := \begin{cases} 1 & \text{falls } A_i \in e_j, \\ 0 & \text{sonst.} \end{cases}$$

Im Beispiel des Graphen in Abbildung 1 ist die Inzidenzmatrix

$\mathrm{IM}_\Gamma =$

e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	
1	0	0	1	1	0	0	0	A_1
1	1	0	0	0	1	0	0	A_2
0	1	1	0	1	0	0	1	A_3
0	0	1	1	0	1	1	0	A_4
0	0	0	0	0	0	1	1	A_5

DEFINITION 1.4. Sei $\Gamma = (V, E)$ ein Graph. Unter einem *Kantenzug* versteht man eine Liste $(A_0, \ldots, A_n)$ von (nicht notwendig verschiedenen) Knoten so daß $\{A_i, A_{i+1}\} \in E$ für $i < n$. Ein Kantenzug heißt *geschlossen*, wenn $A_0 = A_n$. Ein Kantenzug $(A_0, \ldots, A_n)$ heißt ein *Weg*, wenn keine Kante mehrfach auftritt, also wenn $\{A_i, A_{i+1}\} \neq \{A_j, A_{j+1}\}$ für $i \neq j$. Ein geschlossener Weg heißt ein *Zykel* (oder *Zyklus*). Ein Weg $(A_0, \ldots, A_n)$ in einem Graph heißt *einfach*, wenn $A_i \neq A_j$ für alle $i \neq j$, mit eventueller

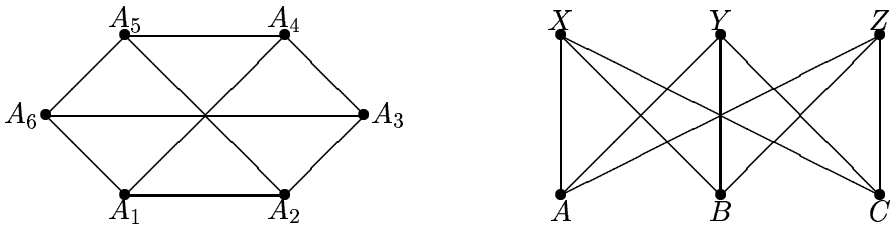


ABBILDUNG 2. Zwei isomorphe Darstellungen des $K33$

Ausnahme der Indexpaare $(0, n)$ und $(n, 0)$. Ein Graph $\Gamma = (V, E)$ heißt *zusammenhängend*, wenn es zu je zwei Knoten $A, B \in V$ einen Kantenzug mit Anfangspunkt A und Endpunkt B gibt.

Man beachte, daß ein Graph $\Gamma = (V, E)$ schon dann zusammenhängend ist, wenn es einen Knoten $A_0 \in V$ gibt, so daß sich jeder andere Knoten $B \in V$ mit A_0 durch einen Kantenzug verbinden läßt.

Aus dem Warshall-Algorithmus erhält man den folgenden *Algorithmus zur Feststellung des Zusammenhangs eines Graphen* Γ: Man stelle die Adjazenzmatrix M_Γ des Graphen auf und bestimme mit Hilfe des Warshall-Algorithmus die reflexiv-transitive Hülle $M_\Gamma^* = \bigvee_{k \geq 0} M_\Gamma^k$ der durch M_Γ gegebenen Relation. Der Graph ist genau dann zusammenhängend, wenn die Matrix M_Γ^* nur aus Einsen besteht. Man beachte, daß dies schon dann der Fall ist, wenn eine Zeile von M_Γ^* nur aus Einsen besteht.

DEFINITION 1.5. Zwei Knoten eines Graphen $\Gamma = (V, E)$ heißen *äquivalent*, wenn sie sich durch einen Kantenzug verbinden lassen. Die Äquivalenzklassen dieser Äquivalenzrelation heißen *Zusammenhangskomponenten* von Γ.

Besteht eine Zusammenhangskomponente eines Graphen nur aus einem einzigen Knoten, so nennt man diesen Knoten einen *isolierten Punkt* des Graphen.

DEFINITION 1.6. Zwei Graphen $\Gamma = (V, E)$ und $\Delta = (V', E')$ heißen *isomorph*, wenn es eine bijektive Abbildung $\varphi\colon V \to V'$ gibt, so daß für die induzierte Abbildung $\bar{\varphi}\colon \mathcal{P}(V) \to \mathcal{P}(V')$ gilt $\bar{\varphi}(E) = E'$.

BEISPIEL 1.7. Man betrachte die beiden Graphen aus Abbildung 2. Beide Graphen sind isomorph, und zwar mittels der Abbildung φ:

$$\begin{array}{ccc} A & \mapsto & A_1 \\ B & \mapsto & A_3 \\ C & \mapsto & A_5 \\ X & \mapsto & A_2 \\ Y & \mapsto & A_4 \\ Z & \mapsto & A_6 \end{array}$$

Hieraus ergibt sich für $\bar{\varphi}$:

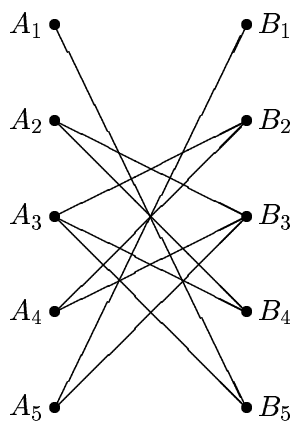
$$\begin{array}{ccccccccc} AX & AY & AZ & BX & BY & BZ & CX & CY & CZ \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ A_1A_2 & A_1A_4 & A_1A_6 & A_3A_2 & A_3A_4 & A_3A_6 & A_5A_2 & A_5A_4 & A_5A_6 \end{array}$$

DEFINITION 1.8. Ein Graph $\Gamma = (V, E)$ heißt *bipartit*, wenn es eine Zerlegung $V = V_1 \cup V_2$ von V in disjunkte Teilmengen V_1 und V_2 gibt so, daß für jede Kante $e \in E$ gilt $e = \{A_1, A_2\}$ mit $A_1 \in V_1$, $A_2 \in V_2$.

BEISPIEL 1.9. Zur graphentheoretischen Behandlung des bekannten Problems vom Fährmann mit Wolf, Ziege und Kohl verwendet man am geeignetsten einen bipartiten Graphen mit den folgenden Eckpunkten:

	V_1	V_2	
A_1	$FWZK \mid$	$\mid FWZK$	B_1
A_2	$FZK \mid W$	$W \mid FZK$	B_2
A_3	$FWK \mid Z$	$Z \mid FWK$	B_3
A_4	$FWZ \mid K$	$K \mid FWZ$	B_4
A_5	$FZ \mid WK$	$WK \mid FZ$	B_5

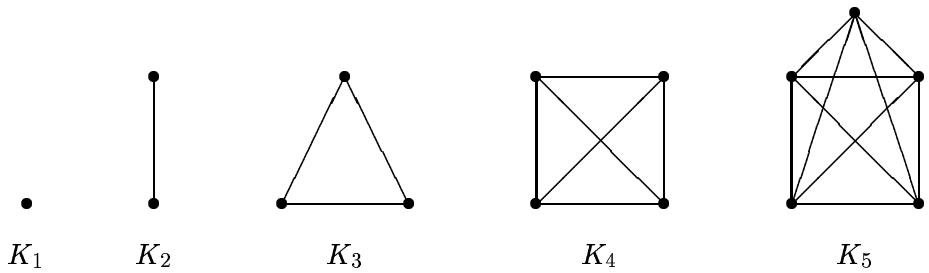
Der folgende Graph beschreibt die Lösung des Problems:



DEFINITION 1.10. Ein Graph $\Gamma = (V, E)$ heißt *vollständig*, wenn E aus *allen* zweielementigen Teilmengen von V besteht. Ein Graph $\Gamma = (V, E)$ heißt *vollständig bipartit*, wenn es eine Zerlegung von V in disjunkte Teilmengen V_1 und V_2 gibt, so daß $E = \{\, \{A_1, A_2\} \mid A_1 \in V_1 \text{ und } A_2 \in V_2 \,\}$.

Der in Abbildung 2 angegebene Garph ist vollständig-bipartit.

Man sieht leicht, daß je zwei vollständige Graphen mit n Eckpunkten isomorph sind. Es gibt also bis auf Isomorphie nur einen vollständigen Graphen mit n Eckpunkten; er wird mit K_n bezeichnet. Die ersten vollständigen Graphen sind



Ebenfalls sind je zwei vollständig-bipartite Graphen mit $\#(V_1) = n$ und $\#(V_2) = m$ isomorph. Der bis auf Isomorphie eindeutig bestimmte vollständig-bipartite Graph wird mit $K_{n,m}$ bezeichnet. In Abbildung 2 ist der $K_{3,3}$ angegeben.

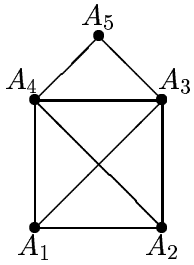
2. Eulersche Wege und Zyklen

In diesem Abschnitt behandeln wir Eulersche und Hamiltonsche Wege und Zyklen. Wir definieren den Begriff des Grads eines Knotens und geben ein hinreichendes und notwendiges Kriterium dafür an, daß Eulersche Wege und Zyklen in einem Graphen existieren. Mit Hilfe der Notwendigkeit des Kriteriums zeigen wir, daß das Königsberger Brückenproblem unlösbar ist.

DEFINITION 2.1. Ein *Eulerscher Weg* in einem Graphen $\Gamma = (V, E)$ ist ein Weg, in dem jede Kante $e \in E$ genau einmal vorkommt. Ein *Eulerscher Zyklus* ist ein Eulerscher Weg, der gleichzeitig ein Zyklus ist.

DEFINITION 2.2. Sei $\Gamma = (V, E)$ ein Graph. Unter dem *Grad* eines Knotens $A \in V$, geschrieben $\deg(A)$, versteht man die Anzahl derjenigen Kanten $e \in E$, die mit A inzidieren, d.h. die A als ein Element besitzen.

BEISPIEL 2.3.



Dann ist

$$\deg(A_1) = \deg(A_2) = 3,\ \deg(A_3) = \deg(A_4) = 4,\ \deg(A_5) = 2.$$

Sei $\Gamma = (V, E)$ ein Graph mit $V = \{A_1, \ldots, A_n\}$ und $M_\Gamma = (a_{ij}) \in M(n \times n, \{0, 1\})$ die Adjazenzmatrix von Γ. Es ist also $a_{ij} = 1$ genau dann, wenn A_i und A_j durch eine Kante verbunden werden. Damit ergibt sich

$$\begin{aligned} \deg(A_i) &= \text{Anzahl der } j \text{ mit } a_{ij} = 1 \\ &= \text{Zeilensumme der } i\text{-ten Zeile von } M_\Gamma. \end{aligned}$$

Ferner gibt es einen entsprechenden Zusammenhang der Gradbegriffs mit der Inzidenzmatrix IM_Γ von Γ. Sei also noch $E = \{e_1, \ldots, e_m\}$ und $\mathrm{IM}_\Gamma = (b_{ij}) \in M(n \times m, \{0, 1\})$ die Inzidenzmatrix von Γ. Dann ist nach Definition der Inzidenzmatrix $b_{ij} = 1$ genau dann, wenn der Knoten A_i mit der Kante e_j inzidiert, also wieder

$$\begin{aligned} \deg(A_i) &= \text{Anzahl der } j \text{ mit } b_{ij} = 1 \\ &= \text{Zeilensumme der } i\text{-ten Zeile von } \mathrm{IM}_\Gamma. \end{aligned}$$

LEMMA 2.4. *Sei $\Gamma = (V, E)$ ein endlicher Graph. Dann gilt*

$$\sum_{A \in V} \deg(A) = 2 \cdot \#(E).$$

Insbesondere ist also die Anzahl der Knoten $A \in V$ mit $\deg(A)$ ungerade eine gerade Zahl.

Beweis. Induktion nach der Anzahl $m = \#(E)$ der Kanten von Γ. *Basis* $m = 0$. Dann besteht der Graph nur aus isolierten Punkten, also $\deg(A) = 0$ für alle $A \in V$. *Schritt* $m - 1 \mapsto m$. Sie Γ' der Graph, der aus Γ durch Wegnahme einer (beliebigen) Kante e entsteht, also $\Gamma' = (V, E \setminus \{e\})$. Für Γ' gilt nach IH

$$\sum \deg_{\Gamma'}(A) = 2 \cdot \#(E \setminus \{e\}) = 2 \cdot \#(E) - 2.$$

Sei $e = \{B, C\}$. Dann ist

$$\begin{aligned} \deg_{\Gamma'}(B) &= \deg_{\Gamma}(B) - 1, \\ \deg_{\Gamma'}(C) &= \deg_{\Gamma}(C) - 1, \\ \deg_{\Gamma'}(P) &= \deg_{\Gamma}(P) \qquad \text{für } P \notin \{B, C\}. \end{aligned}$$

Es folgt

$$\sum \deg_{\Gamma}(A) = \left(\sum \deg_{\Gamma'}(A)\right) + 2$$

und damit die Behauptung. □

Lemma 2.5. (a) *Sei α ein Eulerscher Weg in einem endlichen Graphen $\Gamma = (V, E)$ mit verschiedenem Anfangs- und Endpunkt A und B. Dann gilt*

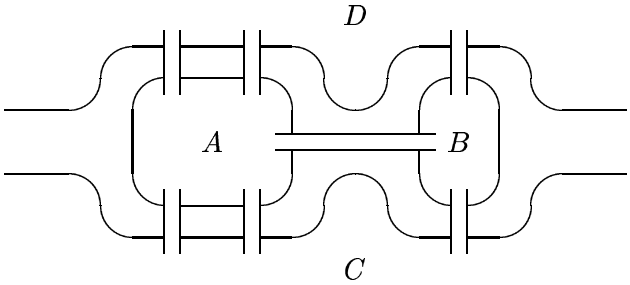
$$\begin{aligned} \deg(A) &\equiv 1 \pmod 2, \\ \deg(B) &\equiv 1 \pmod 2, \\ \deg(P) &\equiv 0 \pmod 2 \quad \textit{für alle } P \in V \setminus \{A, B\}. \end{aligned}$$

(b) *Ist α ein Eulerscher Zyklus in einem endlichen Graphen $\Gamma = (V, E)$, so ist*

$$\deg(P) \equiv 0 \pmod 2 \quad \textit{für alle } P \in V.$$

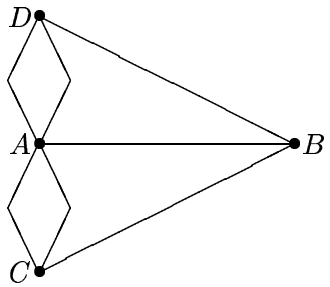
Beweis. (a). Beim Durchlaufen eines Knotens gibt ein innerer Punkt einen Beitrag 2 zum Grad des Knotens, und der Anfangs- und Endpunkt je einen Beitrag 1. Teil (b) ergibt sich mit demselben Argument. □

Beispiel 2.6 (Königsberger Brückenproblem). Zu finden ist ein Spaziergang, der jede der 7 Brücken genau einmal benützt und zum Ausgangspunkt zurückkehrt.



Euler löste dieses Problem im negativen Sinne.

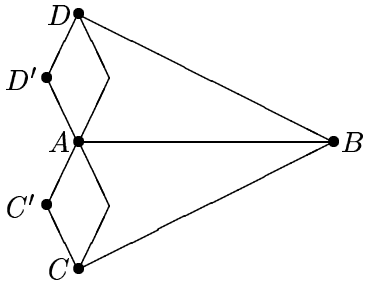
Wir haben folgenden Multi-Graphen Γ (d.h. Graphen mit evtl. mehr als einer Kante zwischen je zwei Knoten) zu betrachten:



Es ist

$$\deg(A) = 5, \quad \deg(B) = \deg(C) = \deg(D) = 3.$$

Durch Einführung zusätzlicher Knoten läßt sich das Problem auf eines für (einfache) Graphen zurückführen. Sei also Γ' der folgende Graph:

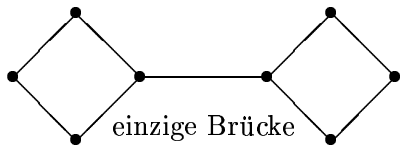


Hier hat man dieselben Grade für A, B, C, D und zusätzlich $\deg(C') = \deg(D') = 2$. Offenbar gibt es genau dann einen Eulerschen Zyklus in Γ', wenn es einen solchen in Γ gibt. Aus dem vorangehenden Lemma folgt aber, daß es in Γ' keinen Eulerschen Zyklus geben kann.

Wir zeigen jetzt, daß das in Lemma 2.5 angegebene notwendige Kriterium für die Existenz Eulerscher Wege bzw. Zykeln auch hinreichend ist.

Definition 2.7. Sei $\Gamma = (V, E)$ ein zusammenhängender Graph. Eine Kante $e \in E$ heißt *Brücke*, wenn der Graph $\Gamma' = (V, E \setminus \{e\})$, der aus Γ durch Wegnahme der Kante e entsteht, unzusammenhängend ist.

Beispiel 2.8.



Lemma 2.9. *Sei $\Gamma = (V, E)$ ein zusammenhängender Graph und $e \in E$. Dann besteht der Graph $\Gamma' = (V, E \setminus \{e\})$ aus höchstens zwei Zusammenhangskomponenten.*

Beweis. Sei $e = \{A, B\}$ mit $A \neq B$. Es genügt zu zeigen, daß in $\Gamma' = (V, E \setminus \{e\})$ jeder Knoten $C \in V$ mit mindestens einem der Knoten A oder B durch einen Kantenzug verbunden werden kann. Sei also $C \in V$. Da Γ zusammenhängend ist, kann in Γ der Knoten C mit A durch einen Kantenzug

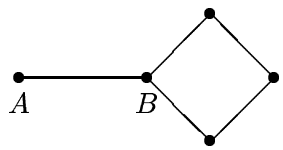
α verbunden werden. Kommt die Kante e in α nicht vor, so sind wir fertig. Andernfalls gibt es eine erstes Auftreten von e in α, d.h. α ist von der Form

$$\alpha = (C, D_1, \ldots, D_n, \underbrace{A, B}_{e}, \ldots, A) \quad \text{oder} \quad \alpha = (C, D_1, \ldots, D_n, \underbrace{B, A}_{e}, \ldots, A).$$

Dann ist aber $\alpha' = (C, D_1, \ldots, D_n, A)$ bzw. $\alpha' = (C, D_1, \ldots, D_n, B)$ ein Kantenzug von C nach A bzw. nach B in Γ. □

DEFINITION 2.10. Sei $e = \{A, B\}$ eine Brücke im Graphen $\Gamma = (V, E)$, so daß der Graph $\Gamma' = (V, E \setminus \{e\})$ den Punkt A als isolierten Punkt besitzt. Dann heißt e *hängende Kante* von Γ und A *hängender Knoten* von Γ.

BEISPIEL 2.11.

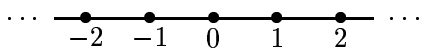


LEMMA 2.12. *Sei $\Gamma = (V, E)$ ein endlicher zusammenhängender Graph mit*

$$\deg(P) \equiv 0 \pmod 2 \quad \textit{für alle } P \in V.$$

Dann besitzt Γ keine Brücke.

Man beachte, daß dies falsch wird, wenn man die Voraussetzung der Endlichkeit fallen läßt. Ein Gegenbeispiel ist der Graph $\Gamma = (\mathbb{Z}, E)$ mit $E = \{\, \{n, n+1\} \mid n \in \mathbb{Z} \,\}$, also



BEWEIS. Sei $e = \{A, B\}$ eine Brücke in Γ. Dann zerfällt $\Gamma' = (V, E \setminus \{e\})$ in zwei Zusammenhangskomponenten. Sei $\Gamma_1 = (V_1, E_1)$ die Zusammenhangskomponente von A. Es gilt dann

$$\begin{aligned} \deg_{\Gamma_1}(A) &= \deg_\Gamma(A) - 1, \\ \deg_{\Gamma_1}(P) &= \deg_\Gamma(P) \quad \text{für alle } P \in V_1 \setminus \{A\}. \end{aligned}$$

Es gibt also in Γ_1 genau einen Knoten (nämlich A) mit ungeradem Grad; alle anderen haben geraden Grad. Dies widerspricht aber Lemma 2.4. □

LEMMA 2.13. *Sei $\Gamma = (V, E)$ ein endlicher zusammenhängender Graph, der genau zwei Knoten mit ungeradem Grad besitzt. Sei $A \in V$ ein Knoten mit $\deg_\Gamma(A)$ ungerade und > 1. Dann gibt es mindestens eine Kante $e = \{A, B\} \in E$, die keine Brücke ist.*

BEWEIS. Sei $e = \{A, B\}$ eine beliebige von A ausgehende Kante. Wenn e keine Brücke ist, so sind wir fertig. Sei also e eine Brücke. Dann zerfällt Γ in zwei Zusammenhangskomponenten. Sei $\Gamma_1 = (V_1, E_1)$ die Zusammenhangskomponente von A. Alle Knoten in Γ_1 haben dann (nach Voraussetzung und Lemma 2.4) einen geraden Grad. Aus dem vorigen Lemma 2.12 folgt, daß Γ_1 keine Brücke besitzt. □

Jetzt können wir zeigen, daß das in Lemma 2.5 angegebene Kriterium für die Existenz Eulerscher Wege und Zykeln auch hinreichend ist.

SATZ 2.14. *Sei $\Gamma = (V, E)$ ein endlicher zusammenhängender Graph.*

(a) *Sind* $A, B \in V$ *Knoten mit*

$$\begin{aligned} \deg(A) &\equiv 1 \pmod 2, \\ \deg(B) &\equiv 1 \pmod 2, \\ \deg(P) &\equiv 0 \pmod 2 \quad \textit{für alle } P \in V \setminus \{A, B\}, \end{aligned}$$

so gibt es in Γ *einen Eulerschen Weg von* A *nach* B.

(b) *Gilt*

$$\deg(P) \equiv 0 \pmod 2 \quad \textit{für alle } P \in V,$$

so besitzt Γ *einen Eulerschen Zyklus.*

Beweis. Induktion über die Anzahl der Kanten von Γ.

(a). *Fall* $\deg(A) = 1$. Dann ist A ein hängender Knoten und die einzige von A ausgehende Kante e ist eine hängende Kante. Wir betrachten den Graphen $\Gamma' = (V \setminus \{A\}, E \setminus \{e\})$; sei $e = \{A, C\}$.

UFall $C = B$. Dann haben alle Knoten in Γ' einen geraden Grad. Nach IH(b) gibt es deshalb in Γ' einen Eulerschen Zyklus mit Anfangs- und Endpunkt B. Zusammengesetzt mit der Kante e liefert dies einen Eulerschen Weg in Γ von A nach B.

UFall $C \neq B$. Dann gilt

$$\begin{aligned} \deg_{\Gamma'}(C) &\equiv 1 \pmod 2, \\ \deg_{\Gamma'}(B) &\equiv 1 \pmod 2, \\ \deg_{\Gamma'}(P) &\equiv 0 \pmod 2 \quad \text{für alle } P \in V \setminus \{A, B, C\}. \end{aligned}$$

Nach IH(a) gibt es deshalb in Γ' einen Eulerschen Weg von C nach B. Zusammengesetzt mit der Kante e liefert dies einen Eulerschen Weg in Γ von A nach B.

Fall $\deg(A) \geq 3$. Dann gibt es nach Lemma 2.13 eine von A ausgehende Kante $e = \{A, C\}$, die keine Brücke ist. Also ist $\Gamma' = (V, E \setminus \{e\})$ zusammenhängend.

UFall $C = B$. Dann haben alle Knoten in Γ' einen geraden Grad. Nach IH(b) gibt es deshalb in Γ' einen Eulerschen Zyklus mit Anfangs- und Endpunkt B. Zusammengesetzt mit der Kante e liefert dies einen Eulerschen Weg in Γ von A nach B.

UFall $C \neq B$. Dann haben in Γ' die Knoten C und B einen ungeraden und alle anderen Knoten (einschließlich A) einen geraden Grad. Nach IH(a) gibt es einen Eulerschen Weg in Γ' von C nach B. Zusammengesetzt mit e liefert dies wieder einen Eulerschen Weg in Γ von A nach B.

(b). Wir können annehmen, daß Γ mindestens eine Kante $e = \{A, B\}$ hat. Dann ist nach Lemma 2.12 diese Kante keine Brücke, also der Graph $\Gamma' = (V, E \setminus \{e\})$ zusammenhängend. Es gilt

$$\begin{aligned} \deg_{\Gamma'}(A) &= \deg_{\Gamma}(A) - 1 \equiv 1 \pmod 2, \\ \deg_{\Gamma'}(B) &= \deg_{\Gamma}(B) - 1 \equiv 1 \pmod 2, \\ \deg_{\Gamma'}(P) &= \deg_{\Gamma}(P) \qquad \equiv 0 \pmod 2 \quad \text{für alle } P \in V \setminus \{A, B\}. \end{aligned}$$

Nach IH(a) gibt es in Γ' einen Eulerschen Weg von A nach B. Zusammengesetzt mit der Kante e liefert dies einen Eulerschen Zyklus in Γ. $\square$

DEFINITION 2.15. Ein *Hamiltonscher Weg* in einem Graphen $\Gamma = (V, E)$ ist ein Weg, in dem jeder Knoten $v \in V$ genau einmal vorkommt. Ein *Hamiltonscher Zyklus* ist ein Hamiltonscher Weg, der gleichzeitig ein Zyklus ist.

Es ist kein effizienter Algorithmus zur Auffindung der Hamiltonschen Wege in einem endlichen Graphen bekannt. Effizient soll hier bedeuten. daß die Komplexität höchstens polynomial in der Anzahl der Kanten ist.

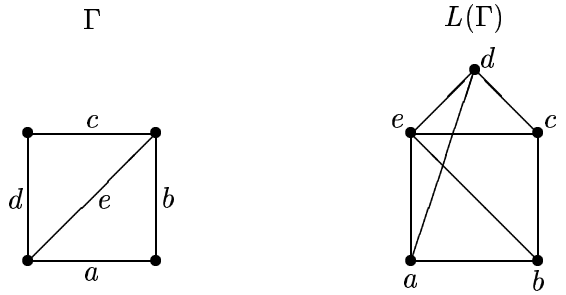
DEFINITION 2.16. Sei $\Gamma = (V, E)$ ein Graph. Unter dem *Kantengraph* (englisch line graph) $L(\Gamma)$ zu Γ versteht man den Graphen

$$L(\Gamma) = (E, E')$$

mit

$$E' := \{\, \{e, f\} \subseteq E \mid e \neq f \text{ und es gibt einen gemeinsamen Eckpunkt } A \text{ von } e \text{ und } f \text{ in } \Gamma \,\}$$

BEISPIEL 2.17.



LEMMA 2.18. *Sei Γ ein Graph, der einen Eulerschen Zyklus besitzt. Dann besitzt der zugehörige Kantengraph $L(\Gamma)$ einen Hamiltonschen Zyklus.*

BEWEIS. Sei

ein Eulerscher Zyklus in Γ. Dann sind nach Definition die Kanten e_i paarweise verschieden. Der zugehörige Hamiltonsche Zyklus in $L(\Gamma)$ verbindet die Knoten $e_1, \ldots, e_n, e_1$ von $L(\Gamma)$. Dies ist ein geschlossener Weg in $L(\Gamma)$, da $\{e_i, e_{i+1}\} \in E'$ und ebenso $\{e_n, e_1\} \in E'$. □

3. Abstände in bewerteten Graphen

Behandelt werden

- der Algorithmus von Moore zur Berechnung der Abstände der Knoten in einem Graphen von einem gegebenen Knoten v_0 (Komplexität $O(n)$),
- eine Verallgemeinerung des Warshall-Algorithmus zur simultanen Bestimmung aller kürzesten Abstände in einem bewerteten Graphen $\Gamma = (V, E, d)$ (Komplexität $O(n^3)$),
- der Algorithmus von Dijkstra zur Bestimmung der kürzesten Abstände von einem festen Knoten v_0 in einem bewerteten Graphen $\Gamma = (V, E, d)$ (Komplexität $O(n^2)$).

Wir beginnen mit einem einfachen Abstandsbegriff. Sei $\Gamma = (V, E)$ ein Graph und $A, B \in V$. Man setzt $d(A, B) := \infty$, wenn A und B in verschiedenen Zusammenhangskomponenten von Γ liegen, und $d(A, B) := k$, wenn A und B durch einen Weg mit k Kanten, aber durch keinen Weg mit weniger Kanten verbunden werden können. Insbesondere hat man also $d(A, A) = 0$ für alle $A \in V$.

Offenbar gilt dann

$$\begin{aligned} &d(A, B) = 0 \quad \text{genau dann, wenn} \quad A = B, \\ &d(A, B) = d(B, A) \quad \text{(Symmetrie)}, \\ &d(A, C) \leq d(A, B) + d(B, C) \quad \text{(Dreiecksungleichung)}. \end{aligned}$$

Wir verwenden sogenannte *Adjazenzmengen* oder *Adjazenzlisten* eines Graphen $\Gamma = (V, E)$. Darunter versteht man die Menge $A_\Gamma(v)$ aller Knoten $w \in V$ mit $\{v, w\} \in E$. Zur Adjazenzmatrix besteht ein einfacher Zusammenhang: Sei $M_\Gamma = (a_{ij}) \in M(n \times n, \{0, 1\})$ die Adjazenzmatrix von Γ (bzgl. einer vorgegebenen Numerierung der Knoten). Dann ist

$$\begin{aligned} A_\Gamma(v_i) &= \{\, v_j \mid a_{ij} = 1 \,\}, \\ \#(A_\Gamma(v_i)) &= \deg_\Gamma(v_i) = \text{Zeilensumme der } i\text{-ten Zeile von } M_\Gamma. \end{aligned}$$

Ferner ist $\sum_{v \in V} \#(A_\Gamma(v)) = \sum \deg_\Gamma(v) = 2 \cdot \#(E)$ nach Lemma 2.4.

Im *Algorithmus von Moore* benötigen wir einen Array $d(v)$, $v \in V$. Am Ende steht in $d(v)$ die Länge des kürzesten Weges von v_0 nach v. Weiter werden Mengen $W(k)$ konstruiert, so daß am Ende gilt $W(k) = \{\, v \in V \mid d(v) = k \,\}$.

```
d(v_0) := 0; d(v) := ∞ für v ∈ V \ {v_0};
W(0) := {v_0};
for k := 1 to n do (n := #(V))
   if W(k − 1) = ∅ then return end;
   W(k) := ∅;
   for v ∈ ⋃{ A(w) | w ∈ W(k − 1) } do
      if d(v) = ∞ then
         d(v) := k;
         W(k) := W(k) ∪ {v};
         end;
      end;
   end;
end;
```

Nach dem k-ten Durchlauf der Schleife sind alle Knoten richtig bestimmt, die von v_0 einen Abstand $\leq k$ haben; dies zeigt man leicht durch Induktion über k. Ferner ist offensichtlich, daß der angegebene Algorithmus von Moore eine Komplexität $O(n)$ hat, also linear ist.

Den restlichen in diesem Abschnitt behandelten Algorithmen liegt ein feinerer Abstandsbegriff zugrunde, bei dem jeder Kante $e \in E$ des betrachteten Graphen $\Gamma = (V, E)$ eine nicht negative rationale Zahl oder ∞ als Abstand $d(e)$ zugeordnet ist.

Definition 3.1. Ein *bewerteter Graph* ist ein Tripel (V, E, d), wobei (V, E) ein (ungerichteter) Graph ist und d eine Funktion $d\colon E \to \mathbb{Q}^+ :=$

$\{ r \in \mathbb{Q} \mid r \geq 0 \}$; d heißt *Bewertungsfunktion*. Wir erweitern d auf alle $\{u, v\} \notin E$ mit $u \neq v$, indem wir setzen

$$d(v_i, v_j) := \infty \quad \text{für } \{v_i, v_j\} \notin E \text{ mit } i \neq j.$$

Die *Länge* eines Weges $\alpha = (v_0, \ldots, v_n)$ ist

$$|\alpha| := \sum_{i<n} d(v_i, v_{i+1}).$$

Um den Warshall-Algorithmus und seine Analyse auf diese Situation verallgemeinern zu können, führen wir zunächst eine Halbringstruktur auf

$$X := \mathbb{Q}^+ \cup \{\infty\} = \{ r \in \mathbb{Q} \mid r \geq 0 \} \cup \{\infty\}$$

ein. $(X, \vee, \wedge)$ wird zu einem Halbring mit Einselement 0 und Nullelement ∞, wenn man $\vee, \wedge \colon X \times X \to X$ definiert durch

$$r \vee s := \min(r, s) := \begin{cases} \text{übl. Min. in } \mathbb{Q} & \text{falls } r, s \in \mathbb{Q} \\ r & \text{falls } r \in \mathbb{Q}, s = \infty \\ s & \text{falls } s \in \mathbb{Q}, r = \infty \\ \infty & \text{falls } r = s = \infty, \end{cases}$$

$$r \wedge s := r + s := \begin{cases} \text{übl. Summe in } \mathbb{Q} & \text{falls } r, s \in \mathbb{Q} \\ \infty & \text{falls } r = \infty \text{ oder } s = \infty. \end{cases}$$

Dies läßt sich leicht beweisen.

Einem bewerteten Graphen $\Gamma = (V, E, d)$ mit $d \colon E \to \mathbb{Q}^+ \cup \{\infty\}$ und $V = \{v_1, \ldots, v_n\}$ ordnen wir eine Matrix $D = (d_{ij}) \in M(n \times n, \mathbb{Q}^+ \cup \{\infty\})$ zu durch

$$d_{ij} := \begin{cases} 0 & \text{für } i = j, \\ d(v_i, v_j) & \text{für } \{v_i, v_j\} \in E, \\ \infty & \text{sonst.} \end{cases}$$

Wir geben zunächst eine graphentheoretische Interpretation der Potenzen D^k der Matrix D über dem Halbring $(\mathbb{Q}^+ \cup \{\infty\}, \min, +)$.

LEMMA 3.2. *Sei D die dem bewerteten Graphen $\Gamma = (V, E, d)$ zugeordnete Matrix und $D^k = (a_{ij})$ die k-te Potenz von D bzgl. der Matrizenmultiplikation über dem Halbring $(\mathbb{Q}^+ \cup \{\infty\}, \min, +)$. Dann ist a_{ij} das Minimum der Längen aller Wege von v_i nach v_j mit höchstens k Kanten ($k \geq 1$).*

BEWEIS. Induktion über k. Der Induktionsanfang $k = 1$ ist trivial. Im Schritt $k - 1 \mapsto k$ haben wir $D^{k-1} = (b_{ij})$ und $D^k = D^{k-1} \cdot D = (a_{ij})$. Dann gilt

$$a_{ij} = \bigvee_{\nu=1,\ldots,n} (b_{i\nu} \wedge d_{\nu j}) = \min_{\nu=1,\ldots,n} (b_{i\nu} + d_{\nu j}).$$

Offensichtlich ist also a_{ij} die Länge des kürzesten Weges α von v_i nach v_j mit höchstens k Kanten. $\square$

LEMMA 3.3. *Der folgende* verallgemeinerte Warshall-Algorithmus *verändert die eben angegebene Matrix D so, daß am Ende der Matrix-Eintrag*

d_{jk} die kürzeste Länge eines Weges von v_j nach v_k ist:

```
for i := 1 to n do
   for j := 1 to n do
      for k := 1 to n do
         d_jk := min(d_jk, d_ji + d_ik)
      end;
   end;
end;
```

Die Komplexität ist $O(n^3)$.

Beweis. Sei $D^{(i)}$ der Inhalt des Speicherplatzes für D nach dem i-ten Durchlauf der äußeren Schleife. Wir zeigen durch Induktion über i, daß $d_{jk}^{(i)}$ die Länge des kürzesten i-Weges (d.h. Weg mit Zwischenpunkten aus $\{v_1, \ldots, v_i\}$) von v_j nach v_k ist.

Nach Annahme ist die Behauptung für $i = 0$ richtig. *Schritt* $i - 1 \mapsto i$: Aufgrund des Algorithmus haben wir

$$d_{jk}^{(i)} = \min(d_{jk}^{(i-1)}, d_{ji}^{(i-1)} + d_{ik}^{(i-1)}).$$

für alle j, i, k. Man betrachte einen i-Weg $\alpha = v_j A_1 \ldots A_n v_k$ von v_j nach v_k, der unter allen solchen i-Wegen kürzeste Länge hat. *Fall* 1. v_i kommt in $A_1 \ldots A_n$ nicht vor. Dann ist

$$\begin{aligned} d_{jk}^{(i)} &= \min(d_{jk}^{(i-1)}, d_{ji}^{(i-1)} + d_{ik}^{(i-1)}) \\ &= d_{jk}^{(i-1)} && \text{nach Annahme} \\ &= |\alpha| && \text{nach IH.} \end{aligned}$$

Fall 2. v_i kommt in $A_1 \ldots A_n$ vor. Wir können annehmen, daß v_i genau einmal in $A_1 \ldots A_n$ vorkommt. Dann hat α die Form

$$\alpha = v_j A_1 \ldots A_{m-1} v_i A_{m+1} \ldots A_n v_j$$

mit $A_1, \ldots, A_{m-1}, A_{m+1}, \ldots, A_n \in \{v_1, \ldots, v_{i-1}\}$, und nach der IH ist dann $d_{ji}^{(i-1)} = |(v_j, A_1, \ldots, A_{m-1}, v_i)|$ und $d_{ik}^{(i-1)} = |(v_i, A_{m+1}, \ldots, A_n, v_k)|$. Es folgt

$$\begin{aligned} d_{jk}^{(i)} &= \min(d_{jk}^{(i-1)}, d_{ji}^{(i-1)} + d_{ik}^{(i-1)}) \\ &= d_{ji}^{(i-1)} + d_{ik}^{(i-1)} && \text{nach Annahme} \\ &= |\alpha| && \text{nach IH.} \end{aligned}$$

Damit haben wir die Korrektheit des verallgemeinerten Warshall-Algorithmus gezeigt; seine Komplexität ist offenbar wieder $O(n^3)$. □

Wir behandeln jetzt den Dijkstra-Algorithmus zur Bestimmung der Abstände von einem festen Knoten. Gegeben ist ein bewerteter Graph $\Gamma = (V, E, w)$ mit $w\colon E \to \mathbb{Q}^+ \cup \{\infty\}$; wir setzen wieder $w(u, v) := \infty$ falls $u \neq v$ und $\{u, v\} \notin E$.

Gesucht ist eine Abbildung $d\colon V \to \mathbb{Q}^+ \cup \{\infty\}$ so daß $d(v)$ die Länge des kürzesten Weges von v_0 nach v ist.

$d(v_0) := 0$; $d(v) := \infty$ für $v \in V \setminus \{v_0\}$;
$U := V$ (= set of undone nodes);
while $U \neq \emptyset$ do
  $\langle$pick $u \in U$ such that $d(u) = \min_{v \in U} d(v)\rangle$;
  for $v \in U$ do
    $d(v) := \min(d(v), d(u) + w(u,v))$;
  end;
  $U := U \setminus \{u\}$;
end;

Die Komplexität ist offenbar $O(n^2)$.

Wir zeigen jetzt die Korrektheit dieses Dijkstra-Algorithmus. Es werden der Reihe nach Knoten $v_0 = u_1, u_2, \ldots, u_n$ aus U entfernt. Für $v \in V$ sei

$\bar{d}(v) :=$ tatsächlicher kürzester Abstand von v_0 nach v,

$d(v) :=$ Ergebnis des Dijkstra-Algorithmus,

$d^{(i)}(v) :=$ Inhalt von $d(v)$ nach dem i-tem Durchlauf der while-Schleife.

Man beachte, daß offenbar gilt

(17) $d^{(0)}(u_i) \geq \cdots \geq d^{(i)}(u_i) = d^{(i+1)}(u_i) = \cdots = d^{(n)}(u_i) = d(u_i) \geq \bar{d}(u_i)$,

(18) $d^{(k)}(u_k) + w(u_k, u_{i+1}) \geq d^{(k)}(u_{i+1})$ für $k \leq i$.

(17) ist klar, und (18) sieht man wie folgt. Im k-ten Durchlauf wird erst u_k ausgewählt und dann für u_k und alle anderen unerledigten Knoten v – insbesondere u_{i+1} – die $d^{(k)}$-Abstände so festgelegt, daß

$$d^{(k)}(v) = \min\{d^{(k-1)}(v),\ d^{(k)}(u_k) + w(u_k, v)\} \leq d^{(k)}(u_k) + w(u_k, v).$$

Wir zeigen

$$\bar{d}(u_i) \geq d(u_i)$$

durch Induktion nach i; wegen $\bar{d}(u_i) \leq d(u_i)$ folgt daraus die Behauptung.

Beweis. *Basis* $i = 1$. Es ist $u_1 = v_0$ und $d(v_0) = \bar{d}(v_0) = 0$. *Schritt* $\{\, k \mid k \leq i \,\} \mapsto i+1$. Sei $u_1 = v_0, v_1, \ldots, v_m, u_{i+1}$ ein kürzester Weg von v_0 nach u_{i+1}.

Fall 1. Alle inneren Knoten $v_1, \ldots, v_m$ liegen in $\{u_1, \ldots, u_i\}$. Sei etwa $v_m = u_k$ mit $k \leq i$. Dann gilt

$$\begin{aligned}
\bar{d}(u_{i+1}) &= \bar{d}(u_k) + w(u_k, u_{i+1}) && \\
&= d(u_k) + w(u_k, u_{i+1}) && \text{nach IH für } k \\
&= d^{(k)}(u_k) + w(u_k, u_{i+1}) && \text{nach (17)} \\
&\geq d^{(k)}(u_{i+1}) && \text{wegen } k \leq i \text{ nach (18)} \\
&\geq d(u_{i+1}) && \text{nach (17).}
\end{aligned}$$

Fall 2. Seien $v_1, \ldots, v_j \in \{u_1, \ldots, u_i\}$, aber $v_{j+1} \notin \{u_1, \ldots, u_i\}$. Sei etwa $v_j = u_k$ mit $k \leq i$. Annahme: $\bar{d}(u_{i+1}) < d(u_{i+1})$. Dann erhält man

$$d(u_{i+1}) > \bar{d}(u_{i+1})$$

$$
\begin{aligned}
&= \bar{d}(u_k) + w(u_k, v_{j+1}) + \dots \\
&= d(u_k) + w(u_k, v_{j+1}) + \dots && \text{nach IH für } k \\
&\geq d(u_k) + w(u_k, v_{j+1}) \\
&= d^{(k)}(u_k) + w(u_k, v_{j+1}) && \text{nach (17)} \\
&\geq d^{(k)}(v_{j+1}) && \text{nach (18) wegen } k \leq i \\
& && \text{und } v_{j+1} \notin \{u_1, \dots, u_i\} \\
&\geq d^{(i+1)}(v_{j+1}) && \text{wegen } k \leq i \text{ nach (17).}
\end{aligned}
$$

Also ist $d(u_{i+1}) = d^{(i+1)}(u_{i+1}) > d^{(i+1)}(v_{j+1})$. Mit $v_{j+1} \notin \{u_1, \dots, u_i\}$ folgt, daß v_{j+1} vor u_{i+1} aus U hätte entnommen werden müssen, und wir haben einen Widerspruch. □

Tabelle 1 enthält ein Beispiel für den Ablauf des Dijkstra-Algorithmus.

4. Ein dem Dijkstra-Algorithmus entsprechender Beweis

Wir wollen jetzt einen Weg aufzeigen, wie man den Dijkstra-Algorithmus aus dem Beweis einer geeigneten Existenzaussage gewinnen kann. Dazu reichern wir den Dijkstra-Algorithmus so an, daß er nicht nur den kürzesten Abstand liefert, sondern zusätzlich noch einen entsprechenden Weg; dies scheint in jedem Fall eine sinnvolle Erweiterung zu sein.

Der im vorangehenden Abschnitt geführte Beweis eignet sich nicht für diesen Zweck, da er wesentlich das Minimumprinzip verwendet und deshalb nicht konstruktiv ist: er begann (im Induktionsschritt) mit „Sei $u_1 = v_0, v_1, \dots, v_m, u_{i+1}$ ein kürzester Weg von v_0 nach u_{i+1}“.

Gegeben ist also ein bewerteter Graph $\Gamma = (V, E, w)$ mit $w\colon E \to \mathbb{N}$. Zur Vereinfachung nehmen wir $V = \{0, 1, \dots, N-1\}$ an, wobei 0 der ausgezeichnete Knoten sein soll. Wir setzen noch $w(0,0) = 0$ und – falls u, v nicht beide 0 sind – $w(u,v) := \infty$ falls $\{u, v\} \notin E$.

Wir zeigen: zu jedem Knoten a gibt es einen „kürzesten Abstand“ $s \in \mathbb{N}$ von dem ausgezeichneten Knoten v_0 zu a. Schreibt man

$$W_f(m) := \sum_{i<m} w(f(i), f(i+1)) \quad \text{(Abstandssumme)},$$

so bedeutet dies

(a) Es gibt eine Folge g von $n+1$ Knoten, die mit dem ausgezeichneten Knoten 0 beginnt und mit dem Knoten a endet, deren Abstandssumme $W_g(n)$ gleich s ist.

(b) Für jede beliebige Folge f von m Knoten, die auch mit dem ausgezeichneten Knoten 0 beginnt und mit dem Knoten a endet, ist die Abstandssumme $W_f(m)$ mindestens so groß wie s.

Mit anderen Worten:

SATZ 4.1. *Für jedes $a \in V$ gibt es ein $s \in \mathbb{N}$ mit*

(a) $(\exists g, n{<}N).g(0) = 0 \wedge g(n) = a \wedge s = W_g(n)$

(b) $\forall f, m.f(0) = 0 \to f(m) = a \to s \leq W_f(m)$

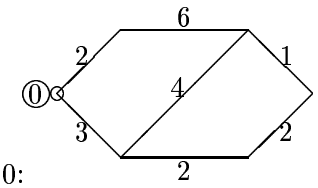
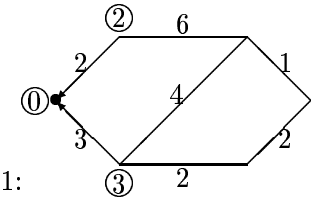
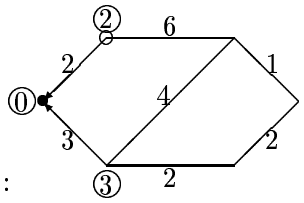
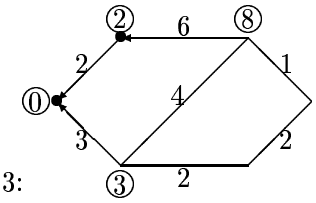
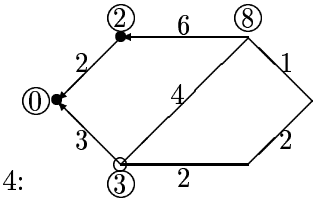
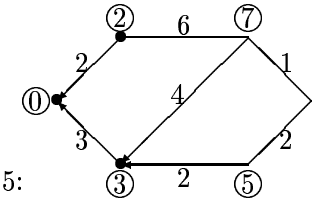
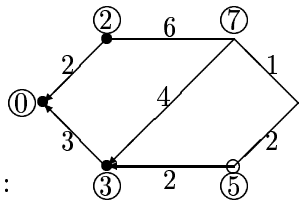
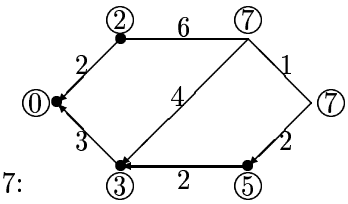
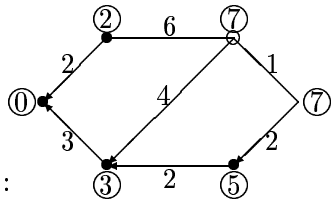
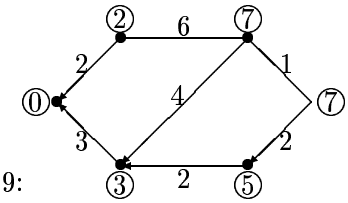
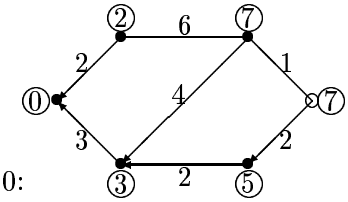
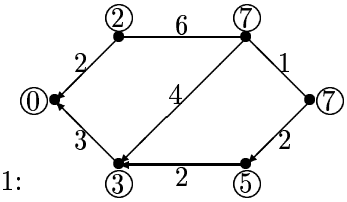
Auswahl	Aktualisierung
 $i = 0$:	 $i = 1$:
 $i = 2$:	 $i = 3$:
 $i = 4$:	 $i = 5$:
 $i = 6$:	 $i = 7$:
 $i = 8$:	 $i = 9$:
 $i = 10$:	 $i = 11$:

TABELLE 1. Ein Beispiel für den Dijkstra-Algorithmus

Zum Beweis werden wir Funktionen $\mathsf{next}\colon V \to V$ und $d\colon V \to \mathbb{N} \cup \{\infty\}$ konstruieren mit folgender intendierter Bedeutung:

$\mathsf{next}(a)$	Nachfolger auf einem kürzestem Weg zu 0, bzw. 0,
$d(a)$	Länge dieses Weges.

Für sie werden wir die folgenden Aussagen beweisen, für alle $a, b \in V$:

$$d(0) = 0 \tag{19}$$

$$a \neq 0 \to \mathsf{next}(a) \neq a \tag{20}$$

$$d(\mathsf{next}(a)) < \infty \tag{21}$$

$$d(a) = d(\mathsf{next}(a)) + w(\mathsf{next}(a), a) \tag{22}$$

$$d(a) \leq d(b) + w(b, a) \tag{23}$$

Wir zeigen zunächst, daß aus (19)–(23) der obige Satz folgt.

Beweis. Setze $s := d(a)$.
(a). Sei

$$h(0) := a, \quad h(i+1) := \begin{cases} \mathsf{next}(h(i)) & \text{falls } h(i) \neq 0 \\ 0 & \text{sonst.} \end{cases}$$

Im Fall $h(i) \neq 0$ ist

$$\begin{aligned} d(h(i+1)) &= d(\mathsf{next}(h(i))) \\ &< d(\mathsf{next}(h(i))) + w(\mathsf{next}(h(i)), h(i)) && \text{wegen (20) und (21)} \\ &= d(h(i)). && \text{wegen (22).} \end{aligned}$$

Es gibt also ein kleinstes $n < N$ mit $h(n) = 0$. Setze $g(i) := h(n-i)$, also $g(0) = h(n) = 0$ und $g(n) = h(0) = a$. Wir zeigen

$$\sum_{k<j} w(g(k), g(k+1)) = d(g(j)).$$

durch Induktion über j. Basis $j = 0$. Klar wegen $g(0) = 0$. Schritt $j \mapsto j+1$. Wegen $j < n$ ist auch $i := n - j - 1 < n$, also

$$\begin{aligned} h(i+1) &= \mathsf{next}(h(i)) \\ h(n-(n-i-1)) &= \mathsf{next}(h(n-(n-i))) \\ g(n-i-1) &= \mathsf{next}(g(n-i)) \\ g(j) &= \mathsf{next}(g(j+1)). \end{aligned}$$

Daraus folgt

$$\begin{aligned} &\sum_{k<j+1} w(g(k), g(k+1)) \\ &= \sum_{k<j} w(g(k), g(k+1)) + w(g(j), g(j+1)) \\ &= d(g(j)) + w(g(j), g(j+1)) && \text{nach IH} \\ &= d(\mathsf{next}(g(j+1))) + w(\mathsf{next}(g(j+1)), g(j+1)) && \text{s. o.} \\ &= d(g(j+1)) && \text{nach (22).} \end{aligned}$$

(b). Gegeben sei $f\colon \mathbb{N} \to V$ mit $f(0) = 0$. Wir zeigen $f(m) = a \to d(a) \leq W_f(m)$ durch Induktion über m. Basis $m = 0$. Dann ist $a = f(0) = 0$, also $d(a) = 0$ nach (19). Schritt $m \mapsto m+1$. Gelte $a = f(m+1)$. Dann ist $s = d(a) = d(f(m+1))$ und

$$\begin{aligned} d(f(m+1)) &\leq d(f(m)) + w(f(m), f(m+1)) && \text{nach (23)} \\ &\leq \sum_{i<m} w(f(i), f(i+1)) + w(f(m), f(m+1)) && \text{nach IH} \\ &= \sum_{i<m+1} w(f(i), f(i+1)) \\ &= W_f(m+1). \end{aligned}$$

Damit ist alles gezeigt. □

Zur Konstruktion der Funktionen $\mathsf{next}\colon V \to V$ und $d\colon V \to \mathbb{N} \cup \{\infty\}$ definieren wir, für jedes $n < N$

- eine Menge set $C_n \subseteq V$, die Menge der *berechneten* Knoten der Stufe n,
- eine Funktion $\mathsf{next}_n\colon V \to V$, die zu jedem Knoten den *nächsten* Knoten auf einem kürzesten Weg von a zu 0 angibt, soweit dies auf der Stufe n bekannt ist,
- eine Funktion $d_n\colon V \to \mathbb{N} \cup \{\infty\}$, die zu jedem Knoten den *Abstand* von 0 auf dem durch die next-Funktion bestimmten Weg zu 0 berechnet, soweit dies auf der Stufe n bekannt ist,
- p_n, der auf der Stufe n ausgewählte („picked“) Knoten.

Sei

$$C_0 := \{0\}, \qquad \mathsf{next}_0(a) := 0, \qquad d_0(a) := w(a, 0).$$

Nehmen wir an, auf der Stufe n haben wir p_n gewählt mit

$$p_n < N \wedge p_n \notin C_n \wedge \forall a. d_n(a) < d_n(p_n) \to a \in C_n. \tag{24}$$

Sei

$$\begin{aligned} C_{n+1} &:= C_n \cup \{p_n\}, \\ \mathsf{next}_{n+1}(a) &:= \begin{cases} p_n & \text{falls } d_n(p_n) + w(p_n, a) < d_n(a), \\ \mathsf{next}_n(a) & \text{sonst,} \end{cases} \\ d_{n+1}(a) &:= \min\{d_n(p_n) + w(p_n, a), d_n(a)\}. \end{aligned}$$

Man beachte, daß der Algorithmus relativ zu einem anderen einfachen Suchalgorithmus pick ist, der zu einer gegebenen Menge $C \subsetneq V$ und Abstandsfunktion $d\colon V \to \mathbb{N} \cup \{\infty\}$ ein $\mathsf{pick}(C, d) \in V \setminus C$ liefert, das unter allen $a \in V \setminus C$ einen minimalen Abstand $d(a)$ hat.

Wir werden für jedes $n < N$ beweisen:

$$\#(C_n) = n + 1, \tag{25}$$

$$d_n(0) = 0, \tag{26}$$

$$a \neq 0 \to \mathsf{next}_n(a) \neq a, \tag{27}$$

$$\mathsf{next}_n(a) \in C_n, \tag{28}$$

$$d_n(\mathsf{next}_n(a)) < \infty, \tag{29}$$

$$d_n(a) = d_n(\mathsf{next}_n(a)) + w(\mathsf{next}_n(a), a), \tag{30}$$

$$b \in C_n \to d_n(a) \leq d_n(b) + w(b, a), \tag{31}$$

$$b \in C_n \to d_n(a) < d_n(b) \to a \in C_n. \tag{32}$$

Aus (25) folgt $C_{N-1} = V$, und (26)–(31) besagen dann gerade, daß $d := d_{N-1}$ und $\mathsf{next} := \mathsf{next}_{N-1}$ die Bedingungen (19)–(23) erfüllen.

Es bleibt zu zeigen, daß (25)–(32) gelten. Wir zeigen dies durch Induktion über n. Der Basisfall $n = 0$ ist klar. Für den Schritt nehmen wir an, daß (25)–(32) für n gelten, und daß p_n entsprechend (24) gewählt ist, also

$$p_n < N, \quad p_n \notin C_n, \quad \forall a. d_n(a) < d_n(p_n) \to a \in C_n.$$

(25) für $n+1$ folgt aus $p_n < N$ und $p_n \notin C_n$. (26) für $n+1$ ist klar, und (27) für $n+1$ folgt aus der Definition mit IH(27). Ebenfalls folgt (28) sofort aus den Definitionen. – Wir zeigen jetzt zunächst

$$d_{n+1}(a) = d_n(a) \quad \text{für alle } a \in C_n. \tag{33}$$

Sei also $a \in C_n$. Wegen $p_n \notin C_n$ folgt $d_n(a) \leq d_n(p_n)$ nach IH(32), also $d_{n+1}(a) = d_n(a)$ nach Definition.

Beweis von $d_{n+1}(\mathsf{next}_{n+1}(a)) < \infty$. Im Fall $d_n(p_n) + w(p_n, a) < d_n(a)$ ist $d_{n+1}(a) = d_n(p_n) + w(p_n, a) < \infty$, also auch

$$d_{n+1}(\mathsf{next}_{n+1}(a)) = d_{n+1}(p_n) = d_n(p_n) < \infty.$$

Andernfalls ist $\mathsf{next}_{n+1}(a) = \mathsf{next}_n(a)$, also wegen $\mathsf{next}_n(a) \in C_n$ nach (33) auch $d_{n+1}(\mathsf{next}_{n+1}(a)) = d_n(\mathsf{next}_n(a)) < \infty$.

Beweis von $\forall a\, d_{n+1}(a) = d_{n+1}(\mathsf{next}_{n+1}(a)) + w(\mathsf{next}_{n+1}(a), a))$. *Fall* 1: $d_n(p_n) + w(p_n, a) < d_n(a)$. Man beachte, daß $d_{n+1}(p_n) = d_n(p_n)$, denn $d_{n+1}(p_n) = \min\{d_n(p_n) + w(p_n, p_n), d_n(p_n)\}$. Also gilt

$$\begin{aligned}
&d_{n+1}(a) \\
&= d_{n+1}(p_n) + w(p_n, a) && \text{nach Definition und Bemerkung} \\
&= d_{n+1}(\mathsf{next}_{n+1}(a)) + w(\mathsf{next}_{n+1}(a), a) && \text{nach Definition von } \mathsf{next}_{n+1}(a).
\end{aligned}$$

Fall 2: Sonst. Dann gilt nach Definition $d_{n+1}(a) = d_n(a)$ und $\mathsf{next}_{n+1}(a) = \mathsf{next}_n(a)$. Die Behauptung folgt aus der IH und (33) wegen $\mathsf{next}_n(a) \in C_n$.

Beweis von $\forall a, b. b \in C_{n+1} \to d_{n+1}(a) \leq d_{n+1}(b) + w(b, a)$. Sei $b \in C_{n+1}$. *Fall* $b \in C_n$. Dann gilt

$$\begin{aligned}
d_{n+1}(a) &\leq d_n(a) && \text{nach Definition} \\
&\leq d_n(b) + w(b, a) && \text{nach IH} \\
&= d_{n+1}(b) + w(b, a) && \text{nach (33).}
\end{aligned}$$

Fall 2: $b = p_n$. Man beachte wieder $d_{n+1}(p_n) = d_n(p_n)$, da $d_{n+1}(p_n) = \min\{d_n(p_n) + w(p_n, p_n), d_n(p_n)\}$. Dann gilt

$$\begin{aligned}
d_{n+1}(a) &\leq d_n(p_n) + w(p_n, a) && \text{nach Definition} \\
&= d_{n+1}(p_n) + w(p_n, a) && \text{nach der Bemerkung} \\
&= d_{n+1}(b) + w(b, a).
\end{aligned}$$

Beweis von $\forall a, b. b \in C_{n+1} \to d_{n+1}(a) < d_{n+1}(b) \to a \in C_{n+1}$. Gelte $b \in C_{n+1}$ und $d_{n+1}(a) < d_{n+1}(b)$. *Fall* $d_n(p_n) + w(p_n, a) < d_n(a)$, d.h.

$d_{n+1}(a) = d_n(p_n) + w(p_n, a)$, also

$$\begin{aligned} d_n(p_n) &\leq d_n(p_n) + w(p_n, a) \\ &= d_{n+1}(a) && \text{nach Definition} \\ &< d_{n+1}(b) && \text{nach Annahme} \\ &\leq d_n(b) && \text{nach Definition.} \end{aligned}$$

Unterfall $b \in C_n$. Dann folgt aus der IH und der obigen Ungleichung $p_n \in C_n$, also ein Widerspruch. *Unterfall* $b = p_n$. Die obige Ungleichung liefert unmittelbar den Widerspruch $d_n(p_n) < d_n(p_n)$.

Fall sonst, d.h. $d_{n+1}(a) = d_n(a)$, also

$$\begin{aligned} d_n(a) &= d_{n+1}(a) && \text{nach Definition} \\ &< d_{n+1}(b) && \text{nach Annahme} \\ &\leq d_n(b) && \text{nach Definition.} \end{aligned}$$

Unterfall $b \in C_n$. Aus der IH folgt dann $a \in C_n$. *Unterfall* $b = p_n$. Die obige Ungleichung liefert $a \in C_n$, nach Wahl von p_n.

KAPITEL 6

Bäume

Ein (ungerichteter) Graph G heißt Baum, wenn er azyklisch und zusammenhängend ist. Dies ist äquivalent damit, daß je zwei Knoten durch einen eindeutig bestimmten Weg verbunden werden können, oder auch damit, daß die Anzahl der Kanten um eins größer als die Anzahl der Knoten ist.

Wir betrachten wieder bewertete Graphen, bei denen jeder Kante eine feste rationale Zahl (oder ∞) als Kantenlänge zuordnet ist. Der Kruskal-Algorithmus bestimmt einen minimalen aufspannenden Baum zu einem gegebenen zusammenhängenden bewerteten Graphen. Im Beweis verwenden wir das Austauschlemma der Graphentheorie, das eine einfache Folgerung aus dem Austauschlemma der linearen Algebra ist.

1. Allgemeine Begriffe

DEFINITION 1.1. Ein Graph heißt *azyklisch*, wenn er keine Zykeln besitzt. Ein *Baum* ist ein zusammenhängender azyklischer Graph. Ein azyklischer Graph heißt ein *Wald*.

Ein Graph ist also genau dann ein Wald, wenn alle seine Zusammenhangskomponenten Bäume sind.

LEMMA 1.2. *Für jeden endlichen zusammenhängenden Graphen* $\Gamma = (V, E)$ *gilt* $\#(E) \geq \#(V) - 1$.

BEWEIS. Induktion über $\#(E) =: n$. *Basis* $n = 0$. Trivial. *Schritt* $n \mapsto n + 1$. Sei e eine Kante, $E' := E \setminus \{e\}$ und $\Gamma' := (V, E')$.

Fall Γ' zusammenhängend. Nach IH ist dann $\#(E') \geq \#(V) - 1$, also auch $\#(E) = \#(E') + 1 \geq \#(V) - 1$.

Fall Γ' nicht zusammenhängend. Dann besteht Γ' nach Kapitel 5, Lemma 2.9 aus höchstens zwei Zusammenhangskomponenten, etwa $\Gamma_1 = (V_1, E_1)$ und $\Gamma_2 = (V_2, E_2)$, und es ist $V = V_1 \cup V_2$ und $E \setminus \{e\} = E_1 \cup E_2$. Auf die beiden Teilgraphen kann man die IH anwenden und erhält $\#(E_i) \geq \#(V_i) - 1$, also $\#(E) = \#(E_1) + \#(E_2) + 1 \geq \#(V_1) + \#(V_2) - 1 = \#(V) - 1$. $\square$

SATZ 1.3. *Für einen endlichen zusammenhängenden Graphen* $\Gamma = (V, E)$ *sind die folgenden Aussagen äquivalent.*

(a) Γ *ist ein Baum (d.h.* Γ *ist azyklisch).*
(b) *Je zwei Knoten in* Γ *sind durch einen eindeutig bestimmten Weg verbindbar.*
(c) $\#(E) = \#(V) - 1$.

BEWEIS. (a) $\Rightarrow$ (b). Gelte (a); wir nehmen an, daß (b) falsch ist. Dann gibt es zwei verschiedene Wege

$$v_1, w_1, w_2, \ldots, w_n, v_2 \quad \text{und} \quad v_1, u_1, u_2, \ldots, u_m, v_2.$$

Sei i_0 maximaler Index mit $w_1 = u_1, w_2 = u_2, \ldots, w_{i_0} = u_{i_0}$. Dann ist

$$v_2, w_n, w_{n-1}, \ldots, w_{i_0} = u_{i_0}, \ldots, u_{m-1}, u_m, v_2$$

ein Zyklus.

(b) $\Rightarrow$ (a). Gelte (b); wir nehmen an, daß (a) falsch ist. Es gibt also einen Zyklus $w_1, w_2, \ldots, w_n = w_1$ mit $n \geq 4$. Dann gibt es aber zwei verschiedene Wege von w_1 nach w_2, nämlich w_1, w_2 und $w_1 = w_n, w_{n-1}, \ldots, w_3, w_2$.

(a) $\Rightarrow$ (c). Sei also $\Gamma = (V, E)$ ein Baum mit $\#(V) = n$. Wir zeigen $\#(E) = n - 1$ durch Induktion über n. Für $n = 1, 2$ ist die Behauptung trivial. *Schritt* $n \mapsto n+1$. Durch Entfernen einer Kante aus $\Gamma = (V, E)$ bilde man $\Gamma' = (V, E \setminus \{e\})$. Dann ist Γ' nicht zusammenhängend, denn andernfalls gäbe es einen Zyklus in Γ. Da Γ zusammenhängend ist, zerfällt Γ' nach Kapitel 5, Lemma 2.9 in zwei Zusammenhangskomponenten $\Gamma_1 = (V_1, E_1)$ und $\Gamma_2 = (V_2, E_2)$, und es ist $V = V_1 \cup V_2$ und $E \setminus \{e\} = E_1 \cup E_2$. Auf die beiden Teilgraphen, die wieder Bäume sind, kann man die IH anwenden und erhält $\#(E_i) = \#(V_i) - 1$, also $\#(E) = \#(E_1) + \#(E_2) + 1 = \#(V_1) + \#(V_2) - 1 = \#(V) - 1$.

(c) $\Rightarrow$ (a). Gelte $\#(E) = \#(V) - 1$. Annahme: Γ ist kein Baum. Dann gibt es einen Zyklus in Γ. Wir entfernen eine Kante e aus diesem Zyklus; sei $\Gamma' := (V, E \setminus \{e\})$. Dann ist Γ' zusammenhängend. Nach Lemma 1.2 gilt $\#(E') \geq \#(V) - 1$, also $\#(E) = \#(E') + 1 \geq \#(V) - 1 + 1 = \#(V)$, also $\#(E) \neq \#(V) - 1$. $\square$

Definition 1.4. Ein *Wurzelbaum* $\Gamma = (V, E, r)$ besteht aus einem Baum (V, E) mit einem ausgezeichneten Knoten r, der *Wurzel* genannt wird.

Man beachte, daß es zu einem gegebenen Baum im allgemeinen mehrere nicht isomorphe Wurzelbäume gibt. Dabei heißen zwei Wurzelbäume $\Gamma = (V, E, r)$ und $\Gamma' = (V', E', r')$ isomorph, wenn es einen Graphisomorphismus $\varphi\colon V \to V'$ gibt (d.h. φ induziert eine Bijektion von E auf E'), so daß $\varphi(r) = r'$.

Definition 1.5. Sei $\Gamma = (V, E, r)$ ein Wurzelbaum.

(a) Für jeden Knoten $v \in V$ wird die *Tiefe* definiert als die Länge des eindeutig bestimmten Weges zur Wurzel minus 1. Unter der *Höhe* oder *Tiefe* $\mathsf{dp}(\Gamma)$ des Wurzelbaums versteht man die maximale Tiefe eines Knotens.

(b) Die *partielle Ordnung* auf Γ ist erklärt durch $v_1 \leq v_2$ genau dann, wenn v_1 auf dem eindeutig bestimmten Weg von der Wurzel nach v_2 liegt.

(c) Ein Knoten $v \in V$ heißt *Blatt*, wenn v maximales Element der in (b) definierten partiellen Ordnung ist. Die übrigen Knoten heißen *innere Knoten*. The *Blätterzahl* (engl. leaf size) $\mathsf{ls}(\Gamma)$ von Γ ist die Anzahl der Blätter des Wurzelbaums.

(d) Sei $v \in V$ ein Knoten. Ein Knoten $v_1 \in V$ heißt *Kind* von v, wenn $v \leq v_1$ und $\{v, v_1\}$ eine Kante des Baums ist. v heißt dann *Elternteil* von v_1. Zwei Knoten heißen *Geschwister* (engl. siblings), wenn sie denselben Elternteil haben. Γ heißt *höchstens k-fach verzweigt* (*k-fach verzweigt*), wenn jeder innere Knoten höchstens k (genau k) Nachfolger hat.

Die *Länge* oder *Größe* eines endlichen Graphen $\Gamma = (V, E)$ ist die Anzahl $\#(V)$ der Knoten des Graphen. Wir schreiben $|\Gamma|$ für die Größe von Γ.

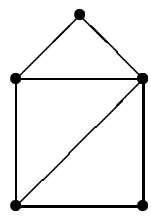
Sei Γ ein höchstens k-fach verzweigter endlicher Wurzelbaum, d.h. jeder Knoten hat höchstens k unmittelbare Nachfolger. Dann gilt offenbar

(a) $|\Gamma| \leq k^{\mathsf{dp}(\Gamma)+1} - 1$ für $k \geq 2$;
(b) $|\Gamma| = 2\mathsf{ls}(\Gamma) - 1$ für genau 2-fach verzweigte Wurzelbäume;
(c) $\mathsf{ls}(\Gamma) \leq |\Gamma|$.

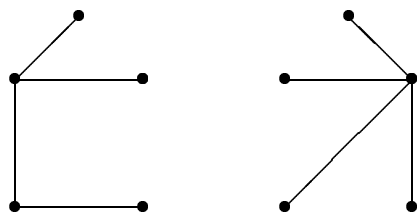
2. Aufspannende Bäume in Graphen: der Kruskal-Algorithmus

DEFINITION 2.1. Sei $\Gamma = (V, E)$ ein zusammenhängender Graph. Ein *aufspannender Baum* von Γ ist ein Baum $T = (V, B)$ mit $B \subseteq E$, der dieselbe Knotenmenge wie Γ hat. Ist $\Gamma = (V, E)$ ein nicht notwendig zusammenhängender Graph, so versteht man unter einem *aufspannenden Wald* von Γ einen Wald $T = (V, B)$ mit $B \subseteq E$, der dieselbe Knotenmenge wie Γ hat.

BEISPIEL 2.2. Man betrachte den Graphen



Aufspannende Bäume sind etwa



DEFINITION 2.3. Sei $\Gamma = (V, E, w)$ ein endlicher zusammenhängender Graph mit Bewertungsfunktion $w\colon E \to \mathbb{Q}^+$. Ein *minimal aufspannender Baum* ist ein aufspannender Baum $T = (V, B)$ mit $w^*(T) := \sum_{b \in B} w(b) \leq w^*(T')$ für alle aufspannenden Bäume $T' = (V, B')$.

Aus dem nächsten Lemma wird folgen, daß der gleich anzugebende Kruskal-Algorithmus einen aufspannenden Baum liefert.

LEMMA 2.4. *Sei $\Gamma = (V, E)$ ein endlicher zusammenhängender Graph und $\mathcal{F} \subseteq \mathcal{P}(E)$ die Menge aller azyklischen Teilmengen von E. Offenbar ist $\mathcal{F}$ durch $\subseteq$ partiell geordnet. Sei $F_0 \in \mathcal{F}$ maximales Element bzgl. $\subseteq$. Dann ist (V, F_0) ein aufspannender Baum.*

BEWEIS. Sei F_0 ein maximales Element und nehmen wir an, daß (V, F_0) kein aufspannender Baum ist. Dann gibt es einen Knoten $v_0 \in V$, der auf keiner Kante von F_0 liegt. Da (V, E) zusammenhängend ist, gibt es eine Kante $e = \{v_0, v\} \in E$ (also $e \notin F_0$). Setze $F_1 := F_0 \cup \{e\}$. F_1 ist azyklisch, denn da F_0 nach Annahme azyklisch ist, müßte ein Zyklus e enthalten und

deshalb läge v_0 doch auf einer Kante von F_0. Damit wäre aber F_0 nicht maximal. $\square$

Wir zeigen jetzt das Austauschlemma der Graphentheorie; es ist eine Folgerung aus dem bekannten Austauschlemma der linearen Algebra.

LEMMA 2.5. *(Austauschlemma der linearen Algebra). Sei V ein Vektorraum über einem Körper K. Sei $v_1, \ldots, v_k$ ein k-Tupel linear unabhängiger Vektoren aus V und $w_1, \ldots, w_l$ ein l-Tupel linear unabhängiger Vektoren aus V mit $l > k$. Dann gibt es einen Index $j \in \{1, \ldots, l\}$ so daß $v_1, \ldots, v_k, w_j$ linear unabhängig ist.*

LEMMA 2.6. *(Austauschlemma der Graphentheorie). Sei $\Gamma = (V, E)$ ein endlicher zusammenhängender Graph. Ferner seien $F_0 \subseteq E$ und $F_1 \subseteq E$ azyklische Teilmengen mit $\#(F_0) < \#(F_1)$. Dann gibt es eine Kante $f \in F_1 \setminus F_0$ so daß $F_0 \cup \{f\}$ azyklisch ist.*

BEWEIS. Sei $V = \{v_1, \ldots, v_n\}$ und $E = \{e_1, \ldots, e_m\}$. Die Inzidenzmatrix $\mathrm{IM}_\Gamma = (a_{ij}) \in M(n \times m, \{0, 1\})$ von Γ ist gegeben durch

$$a_{ij} := \begin{cases} 1 & \text{falls } v_i \in e_j, \\ 0 & \text{sonst.} \end{cases}$$

Die Koeffizienten der Matrix seien als Elemente des Körpers $\mathbb{F}_2 = \{0, 1\}$ betrachtet. Jede Spalte der Matrix $\mathrm{IM}(\Gamma)$ hat genau zwei Komponenten mit dem Wert 1, alle andere Komponenten sind $= 0$.

Wir zeigen, daß ein k-Tupel $e_{j_1}, \ldots, e_{j_k} \in E$ genau dann einen Zyklus enthält, wenn die zugehörigen Spaltenvektoren $s_{j_1}, \ldots, s_{j_k} \in \mathbb{F}_2^n$ linear abhängig sind. Die Behauptung folgt dann aus dem Austauschlemma der linearen Algebra.

$\Longrightarrow$. Nach einer eventuellen Umnumerierung können wir annehmen, daß $e_{j_1}, \ldots, e_{j_p}$ einen Zyklus bilden. Also gibt es Knoten $w_0, w_1, \ldots, w_p = w_0$ mit $e_{j_i} = \{w_{i-1}, w_i\}$. Dann ist aber $s_{j_1} + \ldots + s_{j_p} = 0$, da wir in $\mathbb{F}_2$ arbeiten: In der w_i entsprechenden Zeile von IM_Γ steht im Fall $1 \leq i < p$ genau in den Spalten s_{j_i} und $s_{j_{i+1}}$ eine 1, und im Fall $i = p$ genau in den Spalten s_{j_p} und s_{j_1} eine 1.

$\Longleftarrow$. Sind $s_{j_1}, \ldots, s_{j_k} \in \mathbb{F}_2^n$ linear abhängig, so gibt es eine nichttriviale Linearkombination, die den Nullvektor darstellt. Man wähle eine minimale nichttriviale Darstellung des Nullvektors. Nach eventueller Umnumerierung ist dies $s_{j_1} + \ldots + s_{j_p} = 0$. Dann enthält die Kantenmenge $e_{j_1}, \ldots, e_{j_p}$ einen Zyklus (wegen der Minimalität ist sie sogar ein Zyklus). $\square$

Insbesondere ergibt sich: Sind $F_0, F_1 \subseteq E$ azyklische Teilmengen mit $\#(F_0) = \#(F_1)$, so gibt es zu jeder Kante $e \in F_0$ eine Kante $f \in F_1$ derart daß $(F_0 \setminus \{e\}) \cup \{f\}$ azyklisch ist.

SATZ 2.7. *Der folgende* Kruskal-Algorithmus *bestimmt einen minimalen aufspannenden Baum in einem endlichen zusammenhängenden Graphen $\Gamma = (V, E, w)$ mit Bewertungsfunktion w.*

1. *Ordne die Kanten nach aufsteigendem Gewicht: $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_n)$.*

2. Aufbau des Baumes:

```
F:=0;
for i := 1 to n do
   if (F ∪ {e_i} is acyclic) then
      F := F ∪ {e_i}
   end;
end;
```

BEWEIS. Offenbar erzeugt der Kruskal-Algorithmus eine maximale azyklische Teilmenge von E, und nach Lemma 2.4 ist eine maximale azyklische Teilmenge eines zusammenhängenden Graphen ein aufspannender Baum. Zu zeigen bleibt, daß dieser Baum minimales Gewicht hat, d.h. daß es keinen aufspannenden Baum von kleinerem Gewicht gibt.

Der vom Kruskal-Algorithmus gelieferte Baum habe etwa die Kanten $f_1, \ldots, f_k$ mit $w(f_1) \leq w(f_2) \leq \ldots \leq w(f_k)$. Wir nehmen an, daß es einen anderen aufspannenden Baum mit geringerem Gesamtgewicht gibt. Seine Kanten seien etwa $g_1, \ldots, g_k$ mit $w(g_1) \leq w(g_2) \leq \ldots \leq w(g_k)$. Man beachte, daß man nach Satz 1.3 in beiden Fällen die Kantenzahl $k = \#(V) - 1$ hat. OBdA sei das Gegenbeispiel so gewählt, daß es eine maximale Anzahl gemeinsamer Kanten mit dem Kruskal-Baum besitzt. Sei $k_0 \leq k$ derjenige Index mit

$$f_{k_0} \neq g_{k_0} \text{ und } f_j = g_j \text{ für } j > k_0.$$

Fall $w(f_{k_0}) \leq w(g_{k_0})$. Nach dem Austauschlemma kann man g_{k_0} gegen ein f_i austauschen so daß $g_1, \ldots, g_{k_0-1}, f_i, g_{k_0+1}, \ldots, g_k$ wieder ein aufspannender Baum ist. Es gilt dann $i \leq k_0$ (da man sonst kein k-Tupel hat), also $w(f_i) \leq w(f_{k_0}) \leq w(g_{k_0})$ nach Annahme. Der neue Baum ist also ebenfalls ein Gegenbeispiel, und er hat eine Kante mehr mit dem Kruskal-Baum gemeinsam. Dies ist ein Widerspruch.

Fall $w(g_{k_0}) < w(f_{k_0})$. Nach dem Austauschlemma kann man f_{k_0} gegen ein g_j austauschen so daß $f_1, \ldots, f_{k_0-1}, g_j, f_{k_0+1}, \ldots, f_k$ wieder ein aufspannender Baum ist. Es gilt dann $j \leq k_0$, also $w(g_j) \leq w(g_{k_0}) < w(f_{k_0})$ nach Annahme. Dies widerspricht aber der Wahl der f_i: der Kruskal-Algorithmus hätte schon dieses g_j auswählen müssen.

Insgesamt haben wir also gezeigt, daß es kein solches Gegenbeispiel geben kann. □

BEMERKUNG 2.8. Wie entscheidet man im Kruskal-Algorithmus effizient, ob durch Hinzufügen einer Kante ein Zyklus entsteht? Man beachte dazu, daß bei der Ausführung des Kruskal-Algorithmus durch fortgesetztes Hinzufügen von Kanten stets Wälder entstehen. Sei F ein solcher Zwischenwald, und e eine Kante, die nicht zu F gehört. F besteht aus endlich vielen Zusammenhangskomponenten $T_1, \ldots, T_n$, die Bäume sind. Es gilt dann: $F \cup \{e\}$ enthält einen Zyklus genau dann, wenn beide Eckpunkte von e in demselben T_i liegen.

KAPITEL 7

Induktive Definitionen

Viele der bisher und im folgenden untersuchten Begriffe sind „induktiv definiert"; Beispiele sind Terme und Formeln oder auch Herleitungen. Wir wollen uns jetzt allgemein mit induktiven Definitionen befassen.

1. Fixpunkte

DEFINITION 1.1. Es sei U eine beliebige nicht leere Menge. Eine Abbildung $\Gamma\colon \mathcal{P}(U) \to \mathcal{P}(U)$ heißt *Operator* über U. Γ heißt *monoton* (oder auch *schwach wachsend*), wenn für alle $X, Y \subseteq U$ aus $X \subseteq Y$ folgt $\Gamma(X) \subseteq \Gamma(Y)$.

$$\mathsf{lfp}(\Gamma) := \bigcap \{\, X \subseteq U \mid \Gamma(X) \subseteq X \,\}$$

heißt die durch Γ induktiv definierte Menge; $\mathsf{lfp}(\Gamma)$ ist also der Durchschnitt aller Γ-abgeschlossenen Teilmengen von U. Definitionen dieser Art nennt man (verallgemeinerte) *induktive Definitionen*.

SATZ 1.2. *Sei Γ ein monotoner Operator über einer nicht leeren Menge U. Dann gilt*

(a) *Wenn $\Gamma(X) \subseteq X$, so ist $\mathsf{lfp}(\Gamma) \subseteq X$.*
(b) $\Gamma(\mathsf{lfp}(\Gamma)) = \mathsf{lfp}(\Gamma)$.

Insbesondere ist also $\mathsf{lfp}(\Gamma)$ kleinster Fixpunkt von Γ.

BEWEIS. (a) folgt sofort aus der Definition von $\mathsf{lfp}(\Gamma)$. (b). Aus $\Gamma(X) \subseteq X$ folgt $\mathsf{lfp}(\Gamma) \subseteq X$ nach (a), also $\Gamma(\mathsf{lfp}(\Gamma)) \subseteq \Gamma(X) \subseteq X$ wegen der Monotonie von Γ. Nach Definition von $\mathsf{lfp}(\Gamma)$ haben wir damit $\Gamma(\mathsf{lfp}(\Gamma)) \subseteq \mathsf{lfp}(\Gamma)$. Mit der Monotonie von Γ folgt weiter $\Gamma(\Gamma(\mathsf{lfp}(\Gamma))) \subseteq \Gamma(\mathsf{lfp}(\Gamma))$, also $\mathsf{lfp}(\Gamma) \subseteq \Gamma(\mathsf{lfp}(\Gamma))$ wieder nach Definition von $\mathsf{lfp}(\Gamma)$. $\square$

BEISPIEL 1.3. Sei $0 \in U$ und $S\colon U \to U$ eine beliebige Funktion. Für jede Menge $X \subseteq U$ definieren wir

$$\Gamma(X) := \{0\} \cup \{\, S(v) \mid v \in X \,\}.$$

Offenbar ist Γ monoton, und $\mathsf{lfp}(\Gamma)$ besteht aus den Elementen 0, $S(0)$, $S(S(0))$ ... (die nicht notwendig verschieden sind).

BEISPIEL 1.4. Sei $\mapsto$ eine zweistellige Relation auf U, also $\mapsto \subseteq U \times U$; wir nennen $(U, \mapsto)$ auch ein *Reduktionssystem*. Für $X \subseteq U$ sei

$$\Gamma(X) := \{\, u \mid \forall v.(u \mapsto v) \to v \in X \,\}.$$

Offenbar ist Γ monoton; man nennt $\mathsf{lfp}(\Gamma)$ den *erreichbaren Teil* von $(U, \mapsto)$. Gilt $\mathsf{lfp}(\Gamma) = U$, so heißt die Relation $\mapsto$ *terminierend* oder auch *noethersch*; die inverse Relation $\mapsto^{-1}$ heißt *fundiert*.

Die im Satz 1.2(a) auftretende Prämisse $\Gamma(X) \subseteq X$ ist im Spezialfall des Beispiels 1.3 äquivalent zu

$$\forall u.u = 0 \vee \exists v{\in}X(u = S(v)) \to u \in X,$$

d.h. zu

$$0 \in X \wedge \forall v{\in}X\ S(v) \in X),$$

und die Konklusion ist $\forall u{\in}\mathsf{lfp}(\Gamma)\ u \in X$. Offenbar ist also die Aussage von Satz 1.2(a) eine Art allgemeines Induktionsprinzip. Allerdings haben wir im „Induktionsschritt" nicht ganz die gewünschte Form: statt $\forall v{\in}X\ S(v) \in X$ hätten wir gerne $\forall v{\in}\mathsf{lfp}(\Gamma)\ v \in X \to S(v) \in X$. Dies läßt sich jedoch leicht erreichen. Der folgende Satz formuliert die entsprechende Aussage für den allgemeinen Fall.

Satz 1.5 (Induktionsprinzip für induktive Definitionen).

Wenn $\Gamma(X \cap \mathsf{lfp}(\Gamma)) \subseteq X$, *so ist* $\mathsf{lfp}(\Gamma) \subseteq X$.

Beweis. Wegen $\Gamma(X{\cap}\mathsf{lfp}(\Gamma)) \subseteq \Gamma(\mathsf{lfp}(\Gamma)) = \mathsf{lfp}(\Gamma)$ folgt aus der Prämisse $\Gamma(X \cap \mathsf{lfp}(\Gamma)) \subseteq X \cap \mathsf{lfp}(\Gamma)$. Daraus erhält man $\mathsf{lfp}(\Gamma) \subseteq X \cap \mathsf{lfp}(\Gamma)$ nach Definition von $\mathsf{lfp}(\Gamma)$, also $\mathsf{lfp}(\Gamma) \subseteq X$. □

Man beachte, daß sich wegen $\Gamma(X \cap \mathsf{lfp}(\Gamma)) \subseteq \Gamma(\mathsf{lfp}(\Gamma)) = \mathsf{lfp}(\Gamma)$ die Prämisse des Satzes noch abschwächen läßt zu $\mathsf{lfp}(\Gamma) \cap \Gamma(X \cap \mathsf{lfp}(\Gamma)) \subseteq X$. Dies ist jedoch so offensichtlich, daß wir es nicht in die Formulierung des Satzes aufgenommen haben.

Im Beispiel des erreichbaren Teils einer Relation $\mapsto$ schreibt sich die Prämisse $\Gamma(X \cap \mathsf{lfp}(\Gamma)) \subseteq X$ des Induktionssatzs als

$$\forall u.[\forall v.(u \mapsto v) \to v \in X \cap \mathsf{lfp}(\Gamma)] \to u \in X.$$

Beispiel 1.6. Sei $0 \in U$ und $S\colon U \to U$ sowie $\sup\colon U^{\mathbb{N}} \to U$ beliebige Funktionen. Für jede Menge $X \subseteq U$ definieren wir

$$\Gamma(X) := \{0\} \cup \{\, S(v) \mid v \in X \,\} \cup \{\, \sup(f) \mid \forall n{\in}\mathbb{N}\ f(n) \in X \,\}.$$

Offenbar ist Γ monoton, und $\mathsf{lfp}(\Gamma)$ besteht aus den Elementen

$$0, S(0), S(S(0)) \ldots \omega, S(\omega), S(S(\omega)), \ldots \omega \cdot 2, S(\omega \cdot 2), \ldots$$

(die nicht notwendig verschieden sind). Hierbei steht ω für $\sup(f_0)$, wobei $f_0\colon \mathbb{N} \to U$ definiert ist durch $f_0(n) := S^n(0)$.

2. Approximation von Fixpunkten

Der kleinste Fixpunkt $\mathsf{lfp}(\Gamma)$ des monotonen Operators Γ war sozusagen „von oben" definiert, als Durchschnitt aller Mengen X mit $\Gamma(X) \subseteq X$. Wir wollen jetzt zeigen, daß man ihn auch durch schrittweise Approximation „von unten" erhalten kann. In der hier betrachteten allgemeinen Situation müssen wir dazu eine transfinite Iteration der Approximationsschritte längs der Ordinalzahlen durchführen.

Definition 2.1. Es sei Γ ein monotoner Operator über einer nicht leeren Menge U. Wir definieren durch transfinite Rekursion über alle Ordinalzahlen α:

$$\begin{aligned} \Gamma^0 &:= \emptyset, \\ \Gamma^{\alpha+1} &:= \Gamma(\Gamma^\alpha), \end{aligned}$$

$$\Gamma^\lambda \quad := \bigcup_{\xi<\lambda} \Gamma^\xi,$$

wobei λ für eine Limeszahl steht.

LEMMA 2.2. *Ist Γ ein monotoner Operator über einer nicht leeren Menge U, so gilt für alle Ordinalzahlen α:*
(a) $\Gamma^\alpha \subseteq \Gamma^{\alpha+1}$.
(b) $\Gamma^\alpha \subseteq \mathsf{lfp}(\Gamma)$.

BEWEIS. (a). Durch transfinite Induktion nach α. Der Fall $\alpha = 0$ ist trivial. Im Nachfolgerfall haben wir

$$\Gamma^\alpha = \Gamma(\Gamma^{\alpha-1}) \subseteq \Gamma(\Gamma^\alpha) = \Gamma^{\alpha+1}.$$

Hierbei haben wir die IH und die Monotonie von Γ benutzt. Im Limesfall erhält man

$$\Gamma^\lambda = \bigcup_{\xi<\lambda} \Gamma^\xi \subseteq \bigcup_{\xi<\lambda} \Gamma^{\xi+1} = \bigcup_{\xi<\lambda} \Gamma(\Gamma^\xi) \subseteq \Gamma\big(\bigcup_{\xi<\lambda} \Gamma^\xi\big) = \Gamma^{\lambda+1}.$$

Hierbei haben wieder wir die IH und die Monotonie von Γ benutzt.

(b). Durch transfinite Induktion nach α. Der Fall $\alpha = 0$ ist trivial. Im Nachfolgerfall haben wir nach IH $\Gamma^{\alpha-1} \subseteq \mathsf{lfp}(\Gamma)$. Daraus folgt wegen der Monotonie von Γ

$$\Gamma^\alpha = \Gamma(\Gamma^{\alpha-1}) \subseteq \Gamma(\mathsf{lfp}(\Gamma)) = \mathsf{lfp}(\Gamma).$$

Im Limesfall erhält man für alle $\xi < \lambda$ aus der IH $\Gamma^\xi \subseteq \mathsf{lfp}(\Gamma)$. Daraus folgt

$$\Gamma^\lambda = \bigcup_{\xi<\lambda} \Gamma^\xi \subseteq \mathsf{lfp}(\Gamma).$$

Damit ist der Beweis abgeschlossen. □

Weiter zeigen wir, daß der kleinste Fixpunkt erreicht ist, sobald der nächste Iterationsschritt keine neuen Elemente mehr liefert.

LEMMA 2.3. *Ist Γ ein monotoner Operator über einer nicht leeren Menge U, so gilt für alle Ordinalzahlen α:*

$$\textit{Wenn } \Gamma^\alpha = \Gamma^{\alpha+1}, \textit{ so } \mathsf{lfp}(\Gamma) = \Gamma^\alpha.$$

BEWEIS. Sei $\Gamma^\alpha = \Gamma^{\alpha+1}$, also $\Gamma^\alpha = \Gamma(\Gamma^\alpha)$. Dann ist Γ^α ein Fixpunkt von Γ, also $\mathsf{lfp}(\Gamma) \subseteq \Gamma^\alpha$. Die umgekehrte Inklusion folgt aus Teil (b) des vorangehenden Lemmas. □

Das nächste Satz sagt aus, daß der kleinste Fixpunkt immer nach hinreichend vielen Iterationsschritten erreicht wird.

SATZ 2.4. *Ist Γ ein monotoner Operator über einer nicht leeren Menge U, so gibt es eine Ordinalzahl α mit*

$$\Gamma^\alpha = \mathsf{lfp}(\Gamma).$$

Die kleinste Ordinalzahl α mit $\Gamma^\alpha = \mathsf{lfp}(\Gamma)$ bezeichnet man als die Abschlußzahl *von Γ.*

BEWEIS. Gäbe es kein derartiges α, so hätte man stets $\Gamma^\alpha \subsetneq \Gamma^{\alpha+1}$. Die Abbildung $\mathsf{On} \to \mathcal{P}(U)$, $\alpha \mapsto \Gamma^\alpha$ wäre also injektiv, was aus Kardinalitätsgründen unmöglich ist. □

Wir betrachten jetzt den wichtigen Spezialfall von stetigen Operatoren Γ, mit denen wir es im folgenden ausschließlich zu tun haben werden.

DEFINITION 2.5. Sei U eine nicht leere Menge. Eine Teilmenge $\mathcal{Z} \subseteq \mathcal{P}(U)$ heißt *gerichtet*, wenn für jedes endliche $\mathcal{Z}_0 \subseteq \mathcal{Z}$ ein $X \in \mathcal{Z}$ existiert so daß gilt $Y \subseteq X$ für alle $Y \in \mathcal{Z}_0$. Ein Operator $\Gamma\colon \mathcal{P}(U) \to \mathcal{P}(U)$ heißt *stetig*, wenn für jede gerichtete Teilmenge $\mathcal{Z} \subseteq \mathcal{P}(U)$ gilt

$$\Gamma(\bigcup \mathcal{Z}) = \bigcup\{\, \Gamma(X) \mid X \in \mathcal{Z} \,\}.$$

LEMMA 2.6. *Sei Γ ein Operator über einer nicht leeren Menge U. Ist Γ stetig, so ist Γ auch monoton.*

BEWEIS. Für $X, Y \subseteq U$ mit $X \subseteq Y$ gilt wegen der Stetigkeit von Γ

$$\Gamma(Y) = \Gamma(X \cup Y) = \Gamma(X) \cup \Gamma(Y)$$

und damit $\Gamma(X) \subseteq \Gamma(Y)$. □

Bei stetigen Operatoren bricht die transfinite Approximation des Fixpunkts schon nach ω Schritten ab. Wir haben also in diesem Fall eine überschaubare Charakterisierung des kleinsten Fixpunkts „von unten“ erreicht.

SATZ 2.7. *Sei Γ ein stetiger Operator über einer nicht leeren Menge U. Dann gilt*

$$\mathsf{lfp}(\Gamma) = \Gamma^{\omega}.$$

BEWEIS. Nach Lemma 2.3 genügt es, $\Gamma^{\omega+1} = \Gamma^{\omega}$ zu zeigen.

$$\Gamma^{\omega+1} = \Gamma(\Gamma^{\omega}) = \Gamma(\bigcup_{n<\omega} \Gamma^n) = \bigcup_{n<\omega} \Gamma(\Gamma^n) = \bigcup_{n<\omega} \Gamma^{n+1} = \Gamma^{\omega},$$

wobei wir in der drittletzten Gleichung die Stetigkeit von Γ benutzt haben. □

3. Induktive Definitionen mit endlich vielen Vorgängern

Wir betrachten jetzt induktive Definitionen einer besonders einfachen Art, in der der monotone Operator Γ wie folgt durch eine Relation $R \subseteq U^* \times U$ bestimmt ist: für jede Menge $X \subseteq U$ sei

$$\Gamma_R(X) := \{\, u \in U \mid \exists x{\in}X^*(xRu) \,\}.$$

Offenbar ist Γ_R monoton. Wir schreiben $\mathsf{lfp}(R)$ statt $\mathsf{lfp}(\Gamma_R)$. Die Bedingung $\Gamma_R(X) \subseteq X$ aus Satz 1.2(1) ist dann äquivalent zu

$$\forall u \forall x{\in}X^*.\ xRu \to u \in X.$$

Mengen X mit dieser Eigenschaft heißen *R-abgeschlossen*.

Wir zeigen, daß jeder solche Operator Γ_R stetig ist; nach Satz 2.7 folgt daraus $\mathsf{lfp}(R) = \Gamma_R^{\omega}$.

SATZ 3.1. *Für jede Relation $R \subseteq U^* \times U$ ist Γ_R stetig.*

BEWEIS. Sei $\mathcal{Z} \subseteq \mathcal{P}(U)$ gerichtet. Zu zeigen ist $\Gamma_R(\bigcup \mathcal{Z}) = \bigcup\{\, \Gamma_R(X) \mid X \in \mathcal{Z} \,\}$. $\supseteq$ ist klar wegen der Monotonie von Γ_R. $\subseteq$. Sei $u \in \Gamma_R(\bigcup \mathcal{Z})$. Dann gilt $\exists (u_1, \ldots, u_n){\in}(\bigcup \mathcal{Z})^*((u_1, \ldots, u_n)Ru)$. Man wähle $X_1, \ldots, X_n \in \mathcal{Z}$ mit $u_1 \in X_1, \ldots, u_n \in X_n$. Da $\mathcal{Z}$ gerichtet ist, gibt es ein $X \in \mathcal{Z}$ mit $X_1, \ldots, X_n \subseteq X$. Daher ist $(u_1, \ldots, u_n) \in X^*$, also $u \in \Gamma_R(X)$. □

Für die speziellen Operatoren Γ_R ist es nützlich, die grundlegenden Sätze 1.2 und 1.5 wie folgt umzuformulieren und zusammenzufassen.

SATZ 3.2 (Charakteristische Eigenschaften von $\mathsf{lfp}(R)$). *Für jede Relation $R \subseteq U^* \times U$ gilt*

(a) $[\forall u \forall (u_1, \ldots, u_n) \in \mathsf{lfp}(R)^*.\ (u_1, \ldots, u_n) R u \wedge u_1, \ldots, u_n \in X \to u \in X] \to \forall u \in \mathsf{lfp}(R)(u \in X)$.

(b) $\mathsf{lfp}(R)$ *ist R-abgeschlossen, d.h.* $\forall u \forall x \in \mathsf{lfp}(R)^*.\ xRu \to u \in \mathsf{lfp}(R)$.

(c) $\forall u \in \mathsf{lfp}(R) \exists x \in \mathsf{lfp}(R)^* (xRu)$.

Man beachte, daß Teil (a) dieses Satzes ein Induktionsprinzip ausdrückt. Um zu zeigen, daß alle Elemente $u \in \mathsf{lfp}(R)$ in der Menge X liegen, genügt es, den folgenden „Induktionsschritt“ zu beweisen:

> Sei $(u_1, \ldots, u_n) \in \mathsf{lfp}(R)^*$ ein R-Vorgänger von u (und damit $u \in \mathsf{lfp}(R)$). Sind dann $u_1, \ldots, u_n$ Elemente von X, so auch u.

BEWEIS. (a). Die Prämisse $\Gamma_R(X \cap \mathsf{lfp}(\Gamma_R)) \subseteq X$ des allgemeinen Induktionsprinzips für induktive Definitionen (Satz 1.5) ist zur Prämisse der Behauptung äquivalent.

(b) und (c) sind nur Umformulierungen von $\Gamma_R(\mathsf{lfp}(R)) \subseteq \mathsf{lfp}(R)$ bzw. $\mathsf{lfp}(R) \subseteq \Gamma_R(\mathsf{lfp}(R))$, was nach Satz 1.2(b) gilt. □

Statt „X ist kleinster Fixpunkt von Γ_R“ sagt man meist: Sei X induktiv definiert durch die Klausel

Wenn $x \in X^*$ und xRu, so ist $u \in X$.

Hat insbesondere R die Form $R_1 \cup \cdots \cup R_m$ mit $R_i \subseteq U^{n_i} \times U$, so sagt man auch: Sei X induktiv definiert durch die Klauseln

1. Wenn $u_1, \ldots, u_{n_1} \in X$ und $(u_1, \ldots, u_{n_1}) R_1 u$, so ist $u \in X$.

   ...

m. Wenn $u_1, \ldots, u_{n_m} \in X$ und $(u_1, \ldots, u_{n_m}) R_m u$, so ist $u \in X$.

In vielen Fällen werden die Relationen $R_i \subseteq U^{n_i} \times U$ sogar Funktionen sein, d.h. zu jedem $(u_1, \ldots, u_{n_i}) \in U^{n_i}$ existiert genau ein u mit $(u_1, \ldots, u_{n_i}) R_i u$; man schreibt dann $R_i(u_1, \ldots, u_{n_i})$ für u. In diesem Fall sagt man: Sei X induktiv definiert durch die Klauseln

1. Wenn $u_1, \ldots, u_{n_1} \in X$, so ist $R_1(u_1, \ldots, u_{n_1}) \in X$.

   ...

m. Wenn $u_1, \ldots, u_{n_m} \in X$, so ist $R_m(u_1, \ldots, u_{n_m}) \in X$.

Beispiele dazu sind in dieser Vorlesung in großer Zahl aufgetreten.

4. Rekursion

Oft ist es notwendig, auf einer mittels einer Relation R induktiv definierten Menge eine Funktion durch „Rekursion“ zu erklären. Damit dies möglich wird, müssen wir eine weitere Annahme über R machen.

DEFINITION 4.1. Eine Relation $R \subseteq U^* \times U$ heißt *eindeutig*, wenn es zu jedem $u \in U$ höchstens ein $x \in U^*$ gibt mit xRu.

Der nächste Satz sagt aus, daß rekursive Definitionen im Fall eindeutiger Relationen stets möglich sind.

SATZ 4.2 (Rekursionssatz). *Sei $R \subseteq U^* \times U$ eindeutig und $G\colon \mathsf{lfp}(R) \times V^* \to V$. Dann gibt es genau eine Funktion $F\colon \mathsf{lfp}(R) \to V$ so daß für alle $u_1, \ldots, u_n, u \in \mathsf{lfp}(R)$ mit $(u_1, \ldots, u_n)Ru$ gilt*

$$F(u) = G(u, F(u_1), \ldots, F(u_n)).$$

BEWEIS. *Eindeutigkeit.* Angenommen, es gibt noch eine weitere Funktion $F'\colon \mathsf{lfp}(R) \to V$ so daß für alle $u_1, \ldots, u_n, u \in \mathsf{lfp}(R)$ mit $(u_1, \ldots, u_n)Ru$ ebenfalls gilt

$$F'(u) = G(u, F'(u_1), \ldots, F'(u_n)).$$

Wir zeigen $\forall u{\in}\mathsf{lfp}(R)(F(u) = F'(u))$ durch Induktion „über $\mathsf{lfp}(R)$" oder genauer „nach der induktiven Definition von $\mathsf{lfp}(R)$", also aus Satz 3.2(a) mit $X := \{\, u \in U \mid F(u) = F'(u) \,\}$. Seien also $u_1, \ldots, u_n \in \mathsf{lfp}(R)$ mit $F(u_i) = F'(u_i)$ gegeben und gelte $(u_1, \ldots, u_n)Ru$. Dann folgt

$$F(u) = G(u, F(u_1), \ldots, F(u_n)) = G(u, F'(u_1), \ldots, F'(u_n)) = F'(u).$$

Existenz. Wir werden die gesuchte Funktion induktiv definieren. Sei also eine Relation $S \subseteq (\mathsf{lfp}(R) \times V)^* \times (\mathsf{lfp}(R) \times V)$ definiert durch

$$((u_1, v_1), \ldots, (u_n, v_n))S(u, v) \quad :\Longleftrightarrow \quad (u_1, \ldots, u_n)Ru \wedge v = G(u, v_1, \ldots, v_n).$$

Wir wollen zeigen, daß die durch S induktiv definierte Menge $\mathsf{lfp}(S) \subseteq \mathsf{lfp}(R) \times V$ die gesuchte Funktion ist. Dazu zeigen wir zunächst, daß $\mathsf{lfp}(S)$ überhaupt eine Funktion $\mathsf{lfp}(R) \to V$ ist, also daß

$$\forall u{\in}\mathsf{lfp}(R)\exists! v{\in}V((u, v) \in \mathsf{lfp}(S)),$$

und zwar durch Induktion über $\mathsf{lfp}(R)$. Seien also $u_1, \ldots, u_n \in \mathsf{lfp}(R)$ und $u \in U$ mit $(u_1, \ldots, u_n)Ru$ gegeben. Nach IH können wir annehmen, daß es eindeutig bestimmte $v_1, \ldots, v_n$ gibt mit $(u_i, v_i) \in \mathsf{lfp}(S)$. Zu zeigen ist $\exists! v{\in}V((u, v) \in \mathsf{lfp}(S))$. Die Existenz ist einfach: Sei $v := G(u, v_1, \ldots, v_n)$; dann folgt $(u, v) \in \mathsf{lfp}(S)$ aus der S–Abgeschlossenheit von $\mathsf{lfp}(S)$. Zum Beweis der Eindeutigkeit sei noch $(u, v') \in \mathsf{lfp}(S)$. Nach Satz 3.2(c) existieren dann $(u'_1, v'_1), \ldots, (u'_m, v'_m) \in \mathsf{lfp}(S)$ mit $((u'_1, v'_1), \ldots, (u'_m, v'_m))S(u, v')$, also $(u'_1, \ldots, u'_m)Ru \wedge v' = G(u, v'_1, \ldots, v'_m)$. Aus der vorausgesetzten Eindeutigkeit von R folgt zunächst $m = n$ und $u'_i = u_i$. Dann folgt aus $(u_i, v'_i) \in \mathsf{lfp}(S)$ mit der IH $v_i = v'_i$, also $v' = G(u, v'_1, \ldots, v'_n) = G(u, v_1, \ldots, v_n) = v$.

Zu zeigen bleibt, daß die Funktion $\mathsf{lfp}(S)$ die im Satz formulierten Eigenschaften von F hat. Seien also $u_1, \ldots, u_n, u \in \mathsf{lfp}(R)$ mit $(u_1, \ldots, u_n)Ru$ gegeben. Setze $v_i := \mathsf{lfp}(S)(u_i)$ und $v := G(u, v_1, \ldots, v_n)$. Nach Definition von S gilt dann $((u_1, v_1), \ldots, (u_n, v_n))S(u, v)$, also $(u, v) \in \mathsf{lfp}(S)$ wegen der S-Abgeschlossenheit von $\mathsf{lfp}(S)$ und damit auch $\mathsf{lfp}(S)(u) = v = G(u, \mathsf{lfp}(S)(u_1), \ldots, \mathsf{lfp}(S)(u_n))$. $\square$

Auch für rekursive Definitionen haben wir in dieser Vorlesung Beispiele in großer Zahl behandelt.

KAPITEL 8

Endliche Approximation unendlicher Objekte

1. Σ_1-Formeln und Berechenbarkeit

In Kapitel 3 haben wir die Logik freier Algebren entwickelt. Insbesondere wurden in Abschnitt 1 Terme und in Abschnitt 2 Formeln definiert. Ferner waren rekursive Definitionen möglich.

Als Beispiel und zur Erinnerung betrachten wir den Typ $\mathbf{N}$ der natürlichen Zahlen und den Typ $\mathbf{L}(\mathbf{N})$ der Listen natürlicher Zahlen, mit den Konstruktoren nil und cons. Wir können dann rekursiv definieren die Funktionen hd und tl, die das erste Element bzw. den Rest einer Liste liefern:

$$\begin{aligned} \mathsf{hd}(\mathsf{nil}) &:= 0 & \mathsf{tl}(\mathsf{nil}) &:= \mathsf{nil} \\ \mathsf{hd}(\mathsf{cons}(n,l)) &:= n & \mathsf{tl}(\mathsf{cons}(n,l)) &:= l \end{aligned}$$

Die Längenfunktion lh können wir definieren durch

$$\begin{aligned} \mathsf{lh}(\mathsf{nil}) &:= 0 \\ \mathsf{lh}(\mathsf{cons}(n,l)) &:= \mathsf{lh}(n) + 1 \end{aligned}$$

und die Funktion $(l)_i$, die das i-te Element einer Liste l berechnet, durch

$$\begin{aligned} (\mathsf{nil})_i &:= 0 \\ (\mathsf{cons}(n,l))_i &:= \begin{cases} n & \text{falls } i = 0 \\ (l)_{i-1} & \text{sonst} \end{cases} \end{aligned}$$

Definition 1.1. *Beschränkte Formeln* werden aus Primformeln aufgebaut durch

- Implikation $A \to B$,
- Konjunktion $A \wedge B$,
- beschränkte Allquantifikation
$$\forall n^{\mathbf{N}}.n < M \to A \text{ und } \forall l^{\mathbf{L}(\mathbf{N})}.\mathsf{lh}(l) < M \to A$$
- beschränkte Existenzquantifikation
$$\exists n^{\mathbf{N}}.n < M \wedge A \text{ und } \exists l^{\mathbf{L}(\mathbf{N})}.\mathsf{lh}(l) < M \wedge A.$$

Definition 1.2. Σ_1-*Formeln* werden aus beschränkten Formeln aufgebaut durch (unbeschränkte) Existenzquantifikation über finitäre Algebren, haben also die Form $\exists x_1 \ldots \exists x_n A$ mit einer beschränkten Formel A.

Definition 1.3. Eine partielle Funktion $f\colon \mathbf{N} \to \mathbf{N}$ heißt *berechenbar*, wenn ihr Graph Σ_1-definierbar ist.

Beispiel. Die Fakultätsfunktion ist definiert durch

$$\begin{aligned} 0! &:= 1 \\ (n+1)! &:= (n+1) \cdot n! \end{aligned}$$

Ihr Graph läßt sich durch die folgende Σ_1-Formel definieren:

$$\exists l.\mathsf{lh}(l) = n + 1 \wedge (l)_0 = 1 \wedge (l)_n = m \wedge \forall i.i < \mathsf{lh}(l) \to (l)_{i+1} = (i+1) \cdot (l)_i$$

Wir wollen in diesem Kapitel einen allgemeineren Begriff der Berechenbarkeit entwickeln, in dem nicht nur Zahlen und allgemeiner Elemente finitärer Algebren als Argumente zugelassen sind, sondern auch Funktionen, Funktionale etcetera. Unseren Begriff der Berechenbarkeit stützen wir auf zwei Prinzipien: endliche Approximation und Monotonie.

Eine grundlegende Eigenschaft von Berechnungen ist, daß sie *endlich* sein müssen. In einer Berechnung etwa von $\Phi(\varphi)$ kann also die Argumentfunktion φ nur endlich oft aufgerufen werden, d.h. der Wert – falls definiert – muß schon durch eine endliche Teilfunktion von φ bestimmt sein. Dies nennen wir das Prinzip der endlichen Approximation.

Wir wollen dis Diskussion etwas weiter treiben und die Situation auf einer um eins höheren Typenstufe betrachten. Sei $\mathcal{H}$ ein partielles Funktional vom Typ drei, das Funktionale Φ vom Typ zwei auf natürliche Zahlen abbildet. Nehmen wir an, daß $\mathcal{H}(\Phi)$ für ein gegebenes Φ einen definierten Wert hat. Wieder muß die Berechnung endlich sein. Also kann das Argument Φ nur mit endlich vielen Funktionen φ aufgerufen worden sein. Weiter muß jedes solche φ an Φ in einer endlichen Form übergeben worden sein (explizit, etwa als Menge geordneter Paare). Mit anderen Worten, $\mathcal{H}$ und ebenso jedes ihm übergebene Argument Φ vom Typ zwei muß das Prinzip der endlichen Approximation erfüllen. Dies gilt entsprechend auch für höhere Typen.

Um dieses Prinzip genauer beschreiben zu können, benötigen wir den Begriff einer endlichen Approximation Φ_0 eines Funktionals Φ. Darunter verstehen wir eine endliche Menge X von Paaren (φ_0, n) so daß (i) φ_0 eine endliche Funktion ist, (ii) $\Phi(\varphi_0)$ definiert ist mit dem Wert n, und (iii) falls (φ_0, n) und (φ_0', n') zu X gehören und ferner φ_0 und φ_0' „konsistent“ sind, so gilt $n = n'$. Die wesentliche Idee hier ist, daß Φ als die Vereinigung seiner endlichen Approximationen angesehen werden soll. Mit Hilfe dieses Begriffs der endlichen Approximation formulieren wir das

> *Prinzip den endlichen Approximation.* Ist $\mathcal{H}(\Phi)$ definiert mit dem Wert n, so gibt es eine endliche Approximation Φ_0 von Φ so daß $\mathcal{H}(\Phi_0)$ definiert ist mit dem Wert n.

Das Monotonieprinzip formalisiert die einfache Idee, daß ein definierter Wert einer Berechnung von $\mathcal{H}(\Phi)$ sich bei Erweiterung des Arguments Φ nicht mehr ändert. Der hierfür nötige Begriff der Erweiterung ist wie folgt erklärt: Φ' erweitert Φ, wenn für jedes Datenobjekt (φ_0, n) in Φ ein anderes Datenobjekt (φ_0', n) in Φ' existiert so daß φ_0 die Funktion φ_0' erweitert (man beachte die Kontravarianz!). Das zweite gundlegende Prinzip ist dann das

> *Monotonieprinzip.* Ist $\mathcal{H}(\Phi)$ definiert mit Wert n und ist Φ' eine Erweiterung von Φ, so ist auch $\mathcal{H}(\Phi')$ definiert mit Wert n.

Eine unmittelbare Folgerung aus diesen beiden Prinzipien ist, daß jedes Funktional durch sein Verhalten auf endlichen Approximationen bestimmt ist. Haben nämlich Φ und Φ' dieselben endlichen Approximationen und ist $\mathcal{H}(\Phi)$ definiert mit Wert n, so ist nach dem Prinzip der endlichen Approximation $\mathcal{H}(\Phi_0)$ bereits definiert mit Wert n für eine endliche Approximation

Φ_0 von Φ, also aufgrund des Monotonieprinzips auch $\mathcal{H}(\Phi')$ definiert mit Wert n. Also gilt $\mathcal{H}(\Phi) = \mathcal{H}(\Phi')$, für alle $\mathcal{H}$.

Diese Beobachtung erlaubt es uns, einen abstrakten Berechenbarkeitsbegriff zu formulieren:

> *Effektivitätsprinzip.* Ein Objekt ist genau dann berechenbar, wenn die Menge seiner endlichen Approximationen Σ_1-definierbar ist.

BEMERKUNG. Dies ist ein „externer" Begriff der Berechenbarkeit, und es liegt nahe zu fragen, ob man eine äquivalente, aber „interne" Definition finden kann. Dies ist möglich mit Hilfe von Fixpunktoperatoren, die in diesem Kontext zuerst von PLATEK betrachtet wurden. Das Äquivalenzresultat stammt von PLOTKIN [**6**].

2. Informationssysteme

Wir geben eine abstrakte, axiomatische Formulierung der obigen Prinzipien mit Hilfe von sogenannten Informationssystemen. Die Grundidee von Informationssystemen ist es, einen axiomatischen Rahmen bereitzustellen, innerhalb dessen man Approximationen abstrakter Objekte durch konkrete, endliche beschreiben kann. Wir machen hier keinen Versuch, den Begriff der „Konkretheit" oder „Endlichkeit" zu analysieren, sondern gehen stattdessen von einer beliebigen abzählbaren Menge A von „Daten" oder „Datenobjekten" aus, deren Eigenschaften axiomatisch festgelegt werden. Um aus solchen Datenobjekten abstrakte Objekte bilden zu können, brauchen wir den Begriff der „Konsistenz", der festlegt, wann die Elemente einer endlichen Menge von Datenobjekten miteinander verträglich sind. Wir brauchen ferner den Begriff einer „Folgerungsrelation" zwischen konsistenten Mengen X von Daten und einzelnen Datenobjekten a, der intuitiv ausdrückt, daß die in X enthaltene Information ausreicht, den Informationsbaustein a zu erschließen.

Die folgenden Axiome stammen – bis auf eine geringfügige, auf LARSEN und WINSKEL [**5**] zurückgehende Änderung – von SCOTT [**8**].

DEFINITION 2.1. Ein *Informationssystem* ist eine Struktur $(A, \mathsf{Con}, \vdash)$, wobei A eine abzählbare Menge ist (die *Datenobjekte*), Con eine nichtleere Menge von endlichen Teilmengen von A ist (die *konsistenten* Mengen), und $\vdash$ eine Teilmenge von $\mathsf{Con} \times A$ ist (die *Folgerungsrelation*), so daß gilt

(a) Wenn $X \subseteq Y \in \mathsf{Con}$, so $X \in \mathsf{Con}$;
(b) Wenn $a \in A$, so $\{a\} \in \mathsf{Con}$;
(c) Wenn $X \vdash a$, so $X \cup \{a\} \in \mathsf{Con}$;
(d) Wenn $X \in \mathsf{Con}$ und $a \in X$, so $X \vdash a$;
(e) Wenn $X, Y \in \mathsf{Con}$ und $\forall b{\in}Y(X \vdash b)$ und $Y \vdash c$, so $X \vdash c$.

BEZEICHNUNG. Sei $Y \subseteq^{\mathsf{fin}} A$, d.h. endliche Teilmenge von A. Gilt $X \vdash b$ für alle $b \in Y$, so schreiben wir kurz $X \vdash Y$. Mit x, y, z will bezeichnen wir beliebige Teilmengen von A

LEMMA 2.2. *Sei* $\boldsymbol{A} = (A, \mathsf{Con}, \vdash)$ *ein Informationssystem. Dann gilt*

(a) *Wenn* $X \vdash Y$, *so ist* $X \cup Y \in \mathsf{Con}$;
(b) *Wenn* $X \vdash Y$, $X \subseteq X' \in \mathsf{Con}$ *und* $Y' \subseteq Y$, *so gilt auch* $X' \vdash Y'$;
(c) *Wenn* $X \vdash Y$ *und* $Y \vdash Z$, *so* $X \vdash Z$.

BEWEIS. Übung. □

DEFINITION 2.3. Die *Objekte* oder *Ideale* eines Informationssystems $\boldsymbol{A} = (A, \mathsf{Con}, \vdash)$ sind Teilmengen z von A mit den Eigenschaften

(a) $X \subseteq^{\mathsf{fin}} z$ impliziert $X \in \mathsf{Con}$ (z ist „konsistent“);
(b) $X \subseteq^{\mathsf{fin}} z$ und $X \vdash a$ implizieren $a \in z$ (z ist „deduktiv abgeschlossen“).

Die Menge aller Objekte von $\boldsymbol{A}$ bezeichenen wir mit $|\boldsymbol{A}|$.

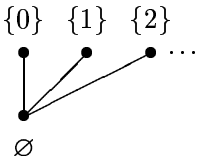
Offenbar ist ein beliebiger Durchschnitt von Idealen wieder ein Ideal.

BEISPIELE 2.4.

(a) Jede abzählbare Menge A läßt sich zu einem *flachen* Informationssystem machen, indem man A als Menge der Datenobjekte auffaßt, und setzt

$$\mathsf{Con} := \{\varnothing\} \cup \{\, \{a\} \mid a \in A \,\} \qquad \text{und} \qquad X \vdash a \;\; :\Longleftrightarrow \;\; a \in X.$$

Für $A = \mathbb{N}$ haben wir das folgende Bild der Con-Mengen:



In diesem Fall sind also die Objekte gerade die Elemente von Con.

(b) Ein besonders wichtiges Beispiel betrifft Approximationen von Funktionen auf einer abzählbaren Menge A mit Werten in einer abzählbaren Menge B. Datenobjekte sind dann die Paare (a, b) mit $a \in A$ und $b \in B$, und

$$\mathsf{Con} := \{\, \{(a_0, b_0), \ldots, (a_{k-1}, b_{k-1})\} \mid \forall i, j . i, j < k \to a_i = a_j \to b_i = b_j \,\},$$
$$X \vdash (a, b) \;\; :\Longleftrightarrow \;\; (a, b) \in X.$$

Man überzeugt sich leicht, daß damit ein Informationssystem definiert wird, dessen Objekte die (Graphen der) partiellen Funktionen von A in B sind.

(c) Sei S eine nicht leere abzählbare Menge. Wir fassen die Elemente der Menge $S^* \cup S^{\mathbb{N}}$ der *Ströme* über S auf als Objekte eines geeigneten Informationssystems $(A, \mathsf{Con}, \vdash)$. Die Menge A der Daten ist S^*, d.h. ein Datenobjekt ist eine endliche Liste von Elementen aus S. $a \sqsubseteq b$ für $a, b \in S^*$ bedeute, daß a ein Anfangsstück von b ist. Offenbar ist $\sqsubseteq$ is reflexiv, transitiv und antisymmetrisch (also eine partielle Ordnung), und erfüllt

$$a, b \sqsubseteq c \to a \sqsubseteq b \vee b \sqsubseteq a. \tag{34}$$

Eine endliche Menge X von Datenobjekten ist konsistent, wenn für alle $a, b \in X$ gilt $a \sqsubseteq b$ oder $b \sqsubseteq a$. Die Folgerungsrelation $X \vdash a$ ist definiert durch $a \sqsubseteq b$ für ein $b \in X$. Man beweist leicht, daß damit ein Informationssystem definiert wird: wir behandeln hier nur Axiom (c) von Definition 2.1, für das (34) benötigt wird. Gelte also $X \vdash a$, d.h. $a \sqsubseteq b$ für ein $b \in X$. Zu zeigen ist $X \cup \{a\} \in \mathsf{Con}$. Sei also $c \in X$; zu zeigen ist dann $a \sqsubseteq c$ oder $c \sqsubseteq a$. *Fall* $c \sqsubseteq b$. Wegen $a, c \sqsubseteq b$ gilt die Behauptung wegen (34). *Fall* $b \sqsubseteq c$. Wegen $a \sqsubseteq b$ erhalten wir $a \sqsubseteq c$. – Statt mit $S^{\mathbb{N}}$ *alle* Folgen von Elementen aus S zu betrachten, kann man sich auch

auf spezielle Folgen aus $S^{\mathbb{N}}$ beschränken und deren Anfangsstücke als Datenobjekte verwenden.

(d) Wir betrachten CAUCHY-Folgen $(q_i)_{i\in\mathbf{N}}$ rationaler Zahlen mit einem speziellen, festen Modul: es soll stets gelten

$$|q_i - q_j| \leq 10^{-i} \text{ für } i < j.$$

Sei $\mathbb{Q}^{\star\mathsf{mod}}$ die Menge der endlichen Anfangsstücke solcher (fest) „modulierter" CAUCHY-Folgen. Wir definieren ein Informationssystem zur Darstellung reeller Funktionen wie folgt. Datenobjekte sind Paare (a, b) mit $a, b \in \mathbb{Q}^{\star\mathsf{mod}}$. Eine Menge $\{(a_0, b_0), \ldots, (a_{k-1}, b_{k-1})\}$ ist konsistent, wenn für alle $i, j < k$ mit $a_i \sqsubseteq a_j$ die Datenobjekte b_i und b_j konsistent sind. Schließlich

$$X \vdash (a, b) \quad :\Longleftrightarrow \quad \text{es gibt ein } (a', b') \in X \text{ mit } a' \sqsubseteq a \text{ und } b \sqsubseteq b'.$$

Man überzeugt sich leicht, daß damit ein Informationssystem definiert wird, dessen Objekte die (Graphen der) reellen Funktionen sind.

DEFINITION 2.5. $(D, \sqsubseteq, \bot)$ heißt *vollständige partielle Ordnung* (englisch *complete partial ordering*, kurz *cpo*), wenn $\sqsubseteq$ eine partielle Ordnung (also reflexiv, transitiv und antisymmetrisch) auf D ist mit kleinstem Element $\bot$, und ferner jede gerichtete Teilmenge $S \subseteq D$ ein Supremum $\bigsqcup S$ in D besitzt.

LEMMA 2.6. *Sei* $\boldsymbol{A} = (A, \mathsf{Con}, \vdash)$ *ein Informationssystem. Dann ist* $(|\boldsymbol{A}|, \subseteq, \varnothing)$ *eine vollständige partielle Ordnung mit Supremumsoperator* $\bigcup$.

BEWEIS. Übung. □

DEFINITION 2.7. Sei $(D, \sqsubseteq, \bot)$ eine vollständige partielle Ordnung. Ein $x \in D$ heißt *kompakt*, wenn es zu jeder gerichteten Teilmenge $S \subseteq D$ mit $x \sqsubseteq \bigsqcup S$ ein $z \in S$ gibt so daß $x \sqsubseteq z$. Die Menge aller kompakten Elemente von D bezeichnen wir mit D_c.

LEMMA 2.8. *Sei* $\boldsymbol{A} = (A, \mathsf{Con}, \vdash)$ *ein Informationssystem. Die kompakten Elemente der vollständigen partiellen Ordnung* $(|\boldsymbol{A}|, \subseteq, \varnothing)$ *lassen sich darstellen in der Form*

$$|\boldsymbol{A}|_c = \{\, \overline{X} \mid X \in \mathsf{Con} \,\},$$

wobei $\overline{X} := \{\, a \in A \mid X \vdash a \,\}$ *der* deduktive Abschluß *von* X *ist.*

BEWEIS. 1. Sei $z \in |\boldsymbol{A}|$ kompakt. Zu zeigen ist $z = \overline{X}$ für ein $X \in \mathsf{Con}$. Das Mengensystem $\{\, \overline{X} \mid X \subseteq^{\mathsf{fin}} z \,\}$ ist gerichtet (denn für $X, Y \subseteq^{\mathsf{fin}} z$ ist $\overline{X \cup Y}$ obere Schranke von $\overline{X}$ und $\overline{Y}$), und es gilt $z \subseteq \bigcup_{X \subseteq^{\mathsf{fin}} z} \overline{X}$. Da z kompakt ist, gilt $z \subseteq \overline{X}$ für ein $X \subseteq^{\mathsf{fin}} z$. Da z auch deduktiv abgeschlossen ist, folgt $\overline{X} \subseteq z$.

2. Sei $X \in \mathsf{Con}$. Zu zeigen ist $\overline{X} \in |\boldsymbol{A}|_c$. Offenbar ist $\overline{X} \in |\boldsymbol{A}|$. Zu zeigen bleibt, daß $\overline{X}$ kompakt ist. Sei also $S \subseteq |\boldsymbol{A}|$ eine gerichtete Teilmenge mit $\overline{X} \subseteq \bigcup S$. Mit $X = \{a_1, \ldots, a_n\}$ haben wir $a_i \in z_i \in S$. Da S gerichtet ist, gibt es ein $z \in S$ mit $z_1, \ldots, z_n \subseteq z$, also $X \subseteq^{\mathsf{fin}} z$ und damit $\overline{X} \subseteq z$. □

DEFINITION 2.9. Eine vollständige partielle Ordnung $(D, \sqsubseteq, \bot)$ heißt *algebraisch*, wenn jedes $x \in D$ das Supremum seiner kompakten Approximationen ist, also

$$x = \bigsqcup \{\, u \in D_c \mid u \sqsubseteq x \,\}.$$

LEMMA 2.10. *Sei* $\boldsymbol{A} = (A, \mathsf{Con}, \vdash)$ *ein Informationssystem. Dann ist* $(|\boldsymbol{A}|, \subseteq, \varnothing)$ *algebraisch.*

BEWEIS. Sei $x \in |\boldsymbol{A}|$. Offenbar ist $x = \bigcup\{\, \overline{X} \mid X \subseteq^{\mathsf{fin}} x \,\}$. □

DEFINITION 2.11. Eine vollständige partielle Ordnung $(D, \sqsubseteq, \bot)$ heißt *Bereich* (auch SCOTT-ERSOV-*Bereich*), wenn

(a) sie algebraisch ist, und
(b) je zwei Elemente $u, v \in D$, die konsistent sind in dem Sinn, daß sie eine obere Schranke $u, v \sqsubseteq w \in D$ besitzen, ein Supremum $u \sqcup v$ in D haben.

Jetzt können wir zeigen, daß die Objekte eines Informationssystems stets einen Bereich bilden:

SATZ 2.12. *Für jedes Informationssystem* $\boldsymbol{A} = (A, \mathsf{Con}, \vdash)$ *ist* $(|\boldsymbol{A}|, \subseteq, \varnothing)$ *ein Bereich mit den kompakten Elementen* $|\boldsymbol{A}|_c = \{\, \overline{X} \mid X \in \mathsf{Con} \,\}$.

BEWEIS. $(|\boldsymbol{A}|, \subseteq, \varnothing)$ ist eine vollständige partielle Ordnung nach Lemma 2.6 und algebraisch nach Lemma 2.10. Seien jetzt $x, y \in |\boldsymbol{A}|$ konsistent sind in dem Sinn, daß sie eine obere Schranke $x, y \subseteq z \in |\boldsymbol{A}|$ besitzen. Dann ist $x \cup y$ das Supremum. Die Charakterisierung der kompakten Elemente wurde in Lemma 2.8 bewiesen. □

BEMERKUNG. Auch die Umkehrung ist richtig: man kann leicht zeigen, daß jeder Bereich sich in der angegebenen Weise als die Menge der Objekte eines geeigneten Informationssystems konstruieren läßt.

3. Abbildungen zwischen Informationssystemen

Wir definieren zu zwei Informationssystemen $\boldsymbol{A}$ und $\boldsymbol{B}$ ein neues Informationssystem $\boldsymbol{A} \to \boldsymbol{B}$, mit dem wir die Funktionen von $\boldsymbol{A}$ nach $\boldsymbol{B}$ approximieren wollen.

DEFINITION 3.1. Seien $\boldsymbol{A} = (A, \mathsf{Con}_A, \vdash_A)$ und $\boldsymbol{B} = (B, \mathsf{Con}_B, \vdash_B)$ Informationssysteme. Die Struktur $\boldsymbol{A} \to \boldsymbol{B} = (C, \mathsf{Con}, \vdash)$ wird wie folgt definiert:

(a) Die Menge C der Datenobjekte ist $C = \mathsf{Con}_A \times B$.
(b) Con besteht aus allen endlichen Mengen $\{(X_1, b_1), \ldots, (X_n, b_n)\} \subseteq C$ derart daß für jede Indexmenge $I \subseteq \{1, \ldots, n\}$ gilt

$$\textstyle\bigcup_{i \in I} X_i \in \mathsf{Con}_A \text{ impliziert } \{\, b_i \mid i \in I \,\} \in \mathsf{Con}_B.$$

(c) Für die Definition der Folgerungsrelation $\vdash$ ist es hilfsreich, zunächst den Begriff der *Anwendung* von $W := \{(X_1, b_1), \ldots, (X_n, b_n)\} \in \mathsf{Con}$ auf $X \in \mathsf{Con}_A$ zu definieren:

$$\{(X_1, b_1), \ldots, (X_n, b_n)\}X := \{\, b_i \mid X \vdash_A X_i \,\}.$$

Aus der Definition von Con ergibt sich, daß diese Menge in Con_B liegt. Wir definieren jetzt $W \vdash (X, b)$ durch $WX \vdash_B b$.

BEMERKUNG. Offenbar ist die Anwendung *monoton im zweiten Argument*, in dem Sinn daß aus $X \vdash_A X'$ folgt ($WX' \subseteq WX$, also auch) $WX \vdash_B WX'$. Die Anwendung ist auch *monoton im ersten Argument*, d.h.

$$W \vdash W' \text{ impliziert } WX \vdash_B W'X.$$

Sei etwa $W = \{ (X_i, b_i) \mid i \in I \}$ und $W' = \{ (X'_j, b'_j) \mid j \in J \}$. Nach Definition ist $W'X = \{ b'_j \mid X \vdash_A X'_j \}$. Sei j fest gewählt mit $X \vdash_A X'_j$; zu zeigen ist $WX \vdash_B b'_j$. Nach Annahme gilt $W \vdash (X'_j, b'_j)$, also $WX'_j \vdash_B b'_j$, also $\{ b_i \mid X'_j \vdash X_i \} \vdash_B b'_j$. Es folgt

$$\begin{aligned} WX &= \{ b_i \mid X \vdash_A X_i \} && \text{nach Definition} \\ &\supseteq \{ b_i \mid X'_j \vdash_A X_i \} && \text{wegen } X \vdash_A X'_j \\ &\vdash_B b'_j && \text{wie eben gezeigt wurde.} \end{aligned}$$

LEMMA 3.2. *Sind $\boldsymbol{A}$ und $\boldsymbol{B}$ Informationssysteme, so auch $\boldsymbol{A} \to \boldsymbol{B}$.*

BEWEIS. Sei $\boldsymbol{A} = (A, \mathsf{Con}_A, \vdash_A)$ und $\boldsymbol{B} = (B, \mathsf{Con}_B, \vdash_B)$. Die Axiome (a), (b) und (d) aus Definition 2.1 sind offenbar erfüllt. Zum Beweis von (c) nehmen wir an, daß

$$\{(X_1, b_1), \ldots, (X_n, b_n)\} \vdash (X, b), \quad \text{d.h.} \quad \{ b_j \mid X \vdash_A X_j \} \vdash_B b.$$

Zu zeigen ist $\{(X_1, b_1), \ldots, (X_n, b_n), (X, b)\} \in \mathsf{Con}$. Sei also $I \subseteq \{1, \ldots, n\}$ und es gelte

$$X \cup \bigcup_{i \in I} X_i \in \mathsf{Con}_A.$$

Zu zeigen ist $b \cup \{ b_i \mid i \in I \} \in \mathsf{Con}_B$. Sei $J \subseteq \{1, \ldots, n\}$ die Menge aller j mit $X \vdash_A X_j$. Dann gilt auch

$$X \cup \bigcup_{i \in I} X_i \cup \bigcup_{j \in J} X_j \in \mathsf{Con}_A.$$

Wegen

$$\bigcup_{i \in I} X_i \cup \bigcup_{j \in J} X_j \in \mathsf{Con}_A,$$

folgt aus der Konsistenz von $\{(X_1, b_1), \ldots, (X_n, b_n)\}$, daß

$$\{ b_i \mid i \in I \} \cup \{ b_j \mid j \in J \} \in \mathsf{Con}_B.$$

Nun gilt aber $\{ b_j \mid j \in J \} \vdash_B b$ nach Annahme. Also

$$\{ b_i \mid i \in I \} \cup \{ b_j \mid j \in J \} \cup \{b\} \in \mathsf{Con}_B.$$

Zum Beweis von (e) nehmen wir an, daß

$$W \vdash \{(X_1, b_1), \ldots, (X_n, b_n)\} \quad \text{und} \quad \{(X_1, b_1), \ldots, (X_n, b_n)\} \vdash (X, b).$$

Zu zeigen ist $W \vdash (X, b)$, also $WX \vdash_B b$. Man erhält

$$\begin{aligned} WX &\vdash_B \bigcup \{ WX_i \mid X \vdash_A X_i \} && \text{(Monotonie im zweiten Argument)} \\ &\vdash_B \{ b_i \mid X \vdash_A X_i \} && \text{wegen } W \vdash (X_i, b_i), \text{ also } WX_i \vdash_B b_i \\ &\vdash_B b && \text{wegen } \{(X_1, b_1), \ldots, (X_n, b_n)\} \vdash (X, b). \end{aligned}$$

Das war zu zeigen. □

Wir geben jetzt eine alternative Charakterisierung des Funktionenraums, und zwar als Menge der „approximierbaren Abbildungen“. Die Idee hierfür ist es, nur solche Abbildungen zwischen $\boldsymbol{A}$ und $\boldsymbol{B}$ zu betrachten, die in einem gewissen Sinn den „Informationsgehalt“ respektieren. Eine solche Abbildung ist gegeben durch eine Relation r zwischen Con_A und B, wobei $r(X, b)$ intuitiv bedeutet, daß immer dann, wenn wir die Information $X \in \mathsf{Con}_A$ über das

Argument zur Verfügung haben, der Wert als Informationsgehalt mindestens das Datenobjekt b haben soll.

DEFINITION 3.3. Seien $\boldsymbol{A} = (A, \mathsf{Con}_A, \vdash_A)$ und $\boldsymbol{B} = (B, \mathsf{Con}_B, \vdash_B)$ Informationssysteme. Eine Relation $r \subseteq \mathsf{Con}_A \times B$ heißt *approximierbare Abbildung*, wenn gilt

(a) Wenn $r(X, b_1), \ldots, r(X, b_n)$, so $\{b_1, \ldots, b_n\} \in \mathsf{Con}_B$;
(b) Wenn $r(X, b_1), \ldots, r(X, b_n)$ und $\{b_1, \ldots, b_n\} \vdash_B b$, so $r(X, b)$;
(c) Wenn $r(X', b)$ und $X \vdash_A X'$, so $r(X, b)$.

Die Schreibweise $r\colon \boldsymbol{A} \to \boldsymbol{B}$ soll bedeuten, daß r eine approximierbare Abbildung von $\boldsymbol{A}$ nach $\boldsymbol{B}$ ist.

SATZ 3.4. *Seien $\boldsymbol{A}$ und $\boldsymbol{B}$ Informationssysteme. Die Objekte von $\boldsymbol{A} \to \boldsymbol{B}$ sind genau die approximierbaren Abbildungen von $\boldsymbol{A}$ nach $\boldsymbol{B}$.*

BEWEIS. Seien $\boldsymbol{A} = (A, \mathsf{Con}_A, \vdash_A)$ und $\boldsymbol{B} = (B, \mathsf{Con}_B, \vdash_B)$. Wenn $r \in |\boldsymbol{A} \to \boldsymbol{B}|$, so ist $r \subseteq \mathsf{Con}_A \times B$ konsistent und deduktiv abgeschlossen. Zu zeigen ist, daß r die Axiome für approximierbare Abbildungen erfüllt.

(a). Gelte $r(X, b_1), \ldots, r(X, b_n)$; zu zeigen ist $\{b_1, \ldots, b_n\} \in \mathsf{Con}_B$. Dies folgt aber offensichtlich aus der Konsistenz von r.

(b). Gelte $r(X, b_1), \ldots, r(X, b_n)$ und $\{b_1, \ldots, b_n\} \vdash_B b$. Zu zeigen ist $r(X, b)$. Wir haben

$$\{(X, b_1), \ldots, (X, b_n)\} \vdash (X, b)$$

nach Definition der Folgerungsrelation $\vdash$ in $\boldsymbol{A} \to \boldsymbol{B}$, also $r(X, b)$ wegen der deduktiven Abgeschlossenheit von r.

(c). Gelte $X \vdash_A X'$ und $r(X', b)$. Zu zeigen ist $r(X, b)$. Wir haben

$$\{(X', b)\} \vdash (X, b)$$

wegen $\{(X', b)\}X = \{b\}$ (dies folgt aus $X \vdash_A X'$), also wieder $r(X, b)$ wegen der deduktiven Abgeschlossenheit von r.

Für die andere Richtung sei $r\colon \boldsymbol{A} \to \boldsymbol{B}$ eine approximierbare Abbildung. Zu zeigen ist $r \in |\boldsymbol{A} \to \boldsymbol{B}|$.

Konsistenz von r. Gelte $r(X_1, b_1), \ldots, r(X_n, b_n)$ und $X = \bigcup\{\, X_i \mid i \in I \,\} \in \mathsf{Con}_A$ für ein $I \subseteq \{1, \ldots, n\}$. Zu zeigen ist $\{\, b_i \mid i \in I \,\} \in \mathsf{Con}_B$. Aus $r(X_i, b_i)$ und $X \vdash_A X_i$ folgt $r(X, b_i)$ nach Axiom (c) für jedes $i \in I$, also $\{\, b_i \mid i \in I \,\} \in \mathsf{Con}_B$ nach Axiom (a).

Deduktive Abgeschlossenheit von r. Gelte $r(X_1, b_1), \ldots, r(X_n, b_n)$ und

$$\{(X_1, b_1), \ldots, (X_n, b_n)\} \vdash (X, b).$$

Zu zeigen ist $r(X, b)$. Nach Definition von $\vdash$ für $\boldsymbol{A} \to \boldsymbol{B}$ haben wir $WX \vdash_B b$ mit $W := \{(X_1, b_1), \ldots, (X_n, b_n)\}$, also $\{\, b_i \mid X \vdash_A X_i \,\} \vdash_B b$. Weiter gilt nach unserer Annahme $r(X_i, b_i)$ auch $r(X, b_i)$ für alle i mit $X \vdash_A X_i$, und zwar nach Axiom (c). Also $r(X, b)$ nach Axiom (b). $\square$

Wir definieren jetzt eine Topologie auf dem Bereich der Objekte eines Informationssystems, geben verschiedene Charakterisierungen ihrer offenen Mengen und untersuchen den zugehörigen Begriff der Stetigkeit. Es wird sich zeigen, daß die bezüglich dieser Topologie stetigen Abbildungen von $|\boldsymbol{A}|$ nach $|\boldsymbol{B}|$ in einer natürlichen Korrespondenz zu den Objekten von $\boldsymbol{A} \to \boldsymbol{B}$ stehen.

Sei $\boldsymbol{A} = (A, \mathsf{Con}, \vdash)$ ein Informationssystem. Wir definieren die Mengen $U_X \subseteq |\boldsymbol{A}|$ durch

$$U_X := \{\, x \in |\boldsymbol{A}| \mid X \subseteq^{\mathsf{fin}} x \,\}.$$

Man beachte, daß wegen der deduktiven Abgeschlossenheit der Objekte $x \in |\boldsymbol{A}|$ für $x \in U_X$ stets gilt $\overline{X} \subseteq x$.

LEMMA 3.5. *Das System aller U_X mit $X \in \mathsf{Con}$ bildet die Basis einer Topologie von $|\boldsymbol{A}|$; sie wird als* SCOTT-*Topologie bezeichnet.*

BEWEIS. Seien $X, Y \in \mathsf{Con}$ und $x \in U_X \cap U_Y$. Zu finden ist ein $Z \in \mathsf{Con}$ mit $x \in U_Z \subseteq U_X \cap U_Y$. Wähle $Z = X \cup Y$. □

LEMMA 3.6. *Sei $\boldsymbol{A}$ ein Informationssystem und $U \subseteq |\boldsymbol{A}|$. Die folgenden Aussagen sind äquivalent:*

(a) *U ist offen in der* SCOTT-*Topologie.*
(b) *U erfüllt*
   - *Wenn $x \in U$ und $x \subseteq y$, so ist $y \in U$ (*ALEXANDROV-*Bedingung).*
   - *Wenn $x \in U$, so ist $\overline{X} \in U$ für ein $X \subseteq^{\mathsf{fin}} x$ (*SCOTT-*Bedingung).*
(c) *$U = \bigcup_{\overline{X} \in U} U_X$.*

Offene Mengen U kann man also ansehen als bestimmt durch ein (möglicherweise unendliches) System endlich beobachtbarer Eigenschaften, nämlich aller X mit $\overline{X} \in U$.

BEWEIS. (a) $\Longrightarrow$ (b). Ist U offen, so ist U die Vereinigung einiger U_X mit $X \in \mathsf{Con}$. Da jedes U_X nach oben abgeschlossen ist, ist es auch U; damit ist die ALEXANDROV-Bedingung bewiesen. Für die SCOTT-Bedingung gelte $x \in U$. Dann ist $x \in U_X \subseteq U$ für ein $X \in \mathsf{Con}$. Man beachte, daß $\overline{X} \in U_X$, also $\overline{X} \in U$, und $X \subseteq^{\mathsf{fin}} x$ wegen $x \in U_X$.

(b) $\Longrightarrow$ (c). $U \subseteq |\boldsymbol{A}|$ erfülle die ALEXANDROV- und SCOTT-Bedingungen. Sei $x \in U$. Nach der SCOTT-Bedingung ist $\overline{X} \in U$ für ein $X \subseteq^{\mathsf{fin}} x$, also $x \in U_X$ für dieses X. Sei umgekehrt $x \in U_X$ für ein $\overline{X} \in U$. Dann gilt $\overline{X} \subseteq x$. Jetzt folgt $x \in U$ aus $\overline{X} \in U$ nach der ALEXANDROV-Bedingung.

(c) $\Longrightarrow$ (a). Die U_X sind die Basismengen der SCOTT-Topologie. □

Wir geben jetzt einige einfache Charakterisierungen der stetigen Funktionen $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$. f heißt *monoton*,wenn aus $x \subseteq y$ folgt $f(x) \subseteq f(y)$.

LEMMA 3.7. *Seien $\boldsymbol{A}$ and $\boldsymbol{B}$ Informationssysteme und $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$. Die folgenden Aussagen sind äquivalent:*

(a) *f ist stetig bezüglich der* SCOTT-*Topologie*
(b) *f ist monoton und erfüllt das „Prinzip der endlichen Approximation“ (Principle of Finite Support, kurz PFS): Wenn $b \in f(x)$, so gilt auch $b \in f(\overline{X})$ für ein $X \subseteq^{\mathsf{fin}} x$.*
(c) *f ist monoton und kommutiert mit gerichteten Vereinigungen: für jede gerichtete Menge $D \subseteq |\boldsymbol{A}|$*

$$f(\bigcup_{x \in D} x) = \bigcup_{x \in D} f(x).$$

Man beachte, daß in (c) die Menge $\{\, f(x) \mid x \in D \,\}$ wegen der Monotonie von f gerichtet ist; also ist ihre Vereinigung ein Objekt in $|\boldsymbol{A}|$. Ferner beachte

man, daß sich aus PFS folgendes ergibt: wenn $Y \subseteq^{\mathsf{fin}} f(x)$, so gilt $Y \subseteq^{\mathsf{fin}} f(\overline{X})$ für ein $X \subseteq^{\mathsf{fin}} x$.

Stetig sind also genau diejenigen Abbildungen $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$, die sich aus der Sicht endlicher Approximationen der abstrakten Objekte $x \in |\boldsymbol{A}|$ und $f(x) \in |\boldsymbol{B}|$ wie folgt beschreiben lassen: Hat man eine endliche Approximation Y des Wertes $f(x)$, so gibt es eine endliche Approximation X des Arguments x derart daß schon $f(\overline{X})$ die Information in Y enthält. Man beachte, daß wegen der Monotonie gilt $f(\overline{X}) \subseteq f(x)$.

Beweis. (a) $\Longrightarrow$ (b). Sei f stetig. Dann ist für jede Basismenge $U_Y \in |\boldsymbol{B}|$ (also $Y \in \mathsf{Con}_B$) die Menge $f^{-1}[U_Y] = \{\, x \mid Y \subseteq^{\mathsf{fin}} f(x) \,\}$ offen in $|\boldsymbol{A}|$. Zum Beweis der Monotonie nehmen wir $x \subseteq y$ an; zu zeigen ist $f(x) \subseteq f(y)$. Sei also $b \in f(x)$, d.h. $\{b\} \subseteq^{\mathsf{fin}} f(x)$. Die offene Menge $f^{-1}[U_{\{b\}}] = \{\, z \mid \{b\} \subseteq^{\mathsf{fin}} f(z) \,\}$ erfüllt die Alexandrov-Bedingung, also aus $x \subseteq y$ können wir schließen auf $\{b\} \subseteq^{\mathsf{fin}} f(y)$, d.h. $b \in f(y)$. Für PFS nehmen wir $b \in f(x)$ an. Die offene Menge $\{\, z \mid \{b\} \subseteq^{\mathsf{fin}} f(z) \,\}$ erfüllt die Scott-Bedingung, also für ein $X \subseteq^{\mathsf{fin}} x$ gilt $\{b\} \subseteq^{\mathsf{fin}} f(\overline{X})$.

(b) $\Longrightarrow$ (a). f sei monoton und erfülle PFS. Zu zeigen ist, daß f stetig ist, d.h. daß für jedes feste $Y \in \mathsf{Con}_B$ die Menge $f^{-1}[U_Y] = \{\, x \mid Y \subseteq^{\mathsf{fin}} f(x) \,\}$ offen ist. Wir zeigen

$$\{\, x \mid Y \subseteq^{\mathsf{fin}} f(x) \,\} = \bigcup \{\, U_X \mid X \in \mathsf{Con}_A \text{ und } Y \subseteq^{\mathsf{fin}} f(\overline{X}) \,\}.$$

Sei $Y \subseteq^{\mathsf{fin}} f(x)$. Nach PFS gilt dann $Y \subseteq^{\mathsf{fin}} f(\overline{X})$ für ein $X \in \mathsf{Con}_A$ mit $X \subseteq^{\mathsf{fin}} x$, und $X \subseteq^{\mathsf{fin}} x$ impliziert $x \in U_X$. Sei umgekehrt $x \in U_X$ für ein $X \in \mathsf{Con}_A$ mit $Y \subseteq^{\mathsf{fin}} f(\overline{X})$. Dann gilt $\overline{X} \subseteq x$, also $Y \subseteq^{\mathsf{fin}} f(x)$ wegen der Monotonie.

Für (b) $\Longleftrightarrow$ (c) nehmen wir, daß f monoton ist. f erfülle PFS, und $D \subseteq |\boldsymbol{A}|$ sei gerichtet. $f(\bigcup_{x \in D} x) \supseteq \bigcup_{x \in D} f(x)$ folgt aus der Monotonie. Für die umgekehrte Inklusion sei $b \in f(\bigcup_{x \in D} x)$. Nach PFS ist $b \in f(\overline{X})$ für ein $X \subseteq^{\mathsf{fin}} \bigcup_{x \in D} x$. Aus der Gerichtetheit von D und der Endlichkeit von X folgt $X \subseteq^{\mathsf{fin}} z$ für ein $z \in D$. Aus $b \in f(\overline{X})$ und der Monotonie folgt $b \in f(z)$. Für die umgekehrte Richtung kommutiere f mit gerichteten Vereinigungen, und es gelte $b \in f(x)$. Dann folgt

$$b \in f(x) = f(\bigcup_{X \subseteq^{\mathsf{fin}} x} \overline{X}) = \bigcup_{X \subseteq^{\mathsf{fin}} x} f(\overline{X}),$$

also $b \in f(\overline{X})$ für ein $X \subseteq^{\mathsf{fin}} x$. □

Satz 3.8. *Seien $\boldsymbol{A}$ und $\boldsymbol{B} = (B, \mathsf{Con}_B, \vdash_B)$ Informationssysteme. Die Objekte von $\boldsymbol{A} \to \boldsymbol{B}$ stehen in einer natürlichen bijektiven Korrespondenz mit den stetigen Funktionen von $|\boldsymbol{A}|$ nach $|\boldsymbol{B}|$, und zwar wie folgt:*

- *Jeder approximierbaren Abbildung $r\colon \boldsymbol{A} \to \boldsymbol{B}$ können wir eine stetige Funktion $|r|\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ zuordnen durch*

$$|r|(z) := \{\, b \in B \mid r(X, b) \text{ für ein } X \subseteq^{\mathsf{fin}} z \,\}.$$

- *Umgekehrt können wir jeder stetigen Funktion $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ eine approximierbare Abbildung $\hat{f}\colon \boldsymbol{A} \to \boldsymbol{B}$ zuordnen durch*

$$\hat{f}(X, b) \quad :\Longleftrightarrow \quad b \in f(\overline{X}).$$

Diese Zuordnungen sind invers zueinander, d.h. es gilt $f = |\hat{f}|$ *und* $r = \widehat{|r|}$.

BEWEIS. Sei r ein Objekt von $\boldsymbol{A} \to \boldsymbol{B}$; nach Satz 3.4 ist dann r eine approximierbare Abbildung. Wir zeigen zunächst, daß $|r|$ wohldefiniert ist. Sei also $z \in |\boldsymbol{A}|$.

$|r|(z)$ ist konsistent: Gegeben seien $b_1, \ldots, b_n \in |r|(z)$. Dann hat man $X_1, \ldots, X_n \subseteq^{\mathsf{fin}} z$ so daß $r(X_i, b_i)$. Also ist $X := X_1 \cup \cdots \cup X_n \subseteq^{\mathsf{fin}} z$ und es gilt $r(X, b_i)$ nach Axiom (c) für approximierbare Abbildungen. Aus Axiom (a) erhalten wir $\{b_1, \ldots, b_n\} \in \mathsf{Con}_B$.

$|r|(z)$ ist deduktiv abgeschlossen: Gegeben seien $b_1, \ldots, b_n \in |r|(z)$ und es gelte $\{b_1, \ldots, b_n\} \vdash_B b$. Zu zeigen ist $b \in |r|(z)$. Wie eben findet man ein $X \subseteq^{\mathsf{fin}} z$ mit $r(X, b_i)$. Axiom (b) impliziert $r(X, b)$, also $b \in |r|(z)$.

Zum Beweis der Stetigkeit von $|r|$ sei $Y \in \mathsf{Con}_B$. Wir müssen zeigen, daß $|r|^{-1}[U_Y]$ offen ist. Nun gilt für jedes $z \in |\boldsymbol{A}|$

$$\begin{aligned}
z \in |r|^{-1}[U_Y] &\iff |r|(z) \in U_Y \\
&\iff \{\, b \in B \mid r(X, b) \text{ für ein } X \subseteq^{\mathsf{fin}} z \,\} \in U_Y \\
&\iff Y \subseteq^{\mathsf{fin}} \{\, b \in B \mid r(X, b) \text{ für ein } X \subseteq^{\mathsf{fin}} z \,\} \\
&\iff \forall b{\in}Y \exists X.\ X \subseteq^{\mathsf{fin}} z \wedge r(X, b) \\
&\iff \forall b{\in}Y \exists X.\ z \in U_X \wedge r(X, b) \\
&\iff z \in \bigcap_{b \in Y} \bigcup \{\, U_X \mid r(X, b) \,\}
\end{aligned}$$

Da Y endlich ist, folgt hieraus, daß $|r|^{-1}[U_Y]$ offen ist.

Sei jetzt $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ stetig. Es ist leicht zu sehen, daß dann $\hat{f}$ eine approximierbare Abbildung ist. Ferner gilt

$$\begin{aligned}
b \in |\hat{f}|(z) &\iff \hat{f}(X, b) && \text{für ein } X \subseteq^{\mathsf{fin}} z \\
&\iff b \in f(\overline{X}) && \text{für ein } X \subseteq^{\mathsf{fin}} z \\
&\iff b \in f(z) && \text{wegen der Monotonie und PFS.}
\end{aligned}$$

Schließlich gilt für jede approximierbare Abbildung $r\colon \boldsymbol{A} \to \boldsymbol{B}$

$$\begin{aligned}
r(X, b) &\iff \exists Y \subseteq^{\mathsf{fin}} \overline{X}\ r(Y, b) && \text{nach Axiom (c)} \\
&\iff b \in |r|(\overline{X}) \\
&\iff \widehat{|r|}(X, b),
\end{aligned}$$

also $r = \widehat{|r|}$. □

BEMERKUNG. Im folgenden schreiben wir meist $r(z)$ für $|r|(z)$, und auch $f(X, b)$ für $\hat{f}(X, b)$.

4. Kartesische Produkte

Wir definieren ein Informationssystem $\boldsymbol{A} \times \boldsymbol{B} = (C, \mathsf{Con}, \vdash)$ aus $\boldsymbol{A}$ und $\boldsymbol{B}$ so, daß die Inklusionsordnung $\subseteq$ auf den Objekten von $|\boldsymbol{A} \times \boldsymbol{B}|$ isomorph ist zu dem kartesischen Produkt der Inklusionsordnungen auf $|\boldsymbol{A}|$ und $|\boldsymbol{B}|$. Ferner zeigen wir, daß die Informationssysteme zusammen mit den stetigen Abbildungen zwischen ihnen eine „kartesisch abgeschlossene Kategorie“ bilden, in der sich auch kleinste Fixpunkte bilden lassen.

Ohne Beschränkung der Allgemeinheit können wir annehmen, daß A und B disjunkt sind. Dann läßt sich jedes Paar (x, y) von Objekten $x \in |\boldsymbol{A}|$ und $y \in |\boldsymbol{B}|$ in jeder Komponente separat approximieren. Als Menge der Datenobjekte in $\boldsymbol{A} \times \boldsymbol{B}$ können wir deshalb die Vereinigung $C := A \cup B$ wählen. Die Begriffe der Konsistenz und der Folgerungsrelation übertragen sich in der erwarteten Weise von $\boldsymbol{A}$ und $\boldsymbol{B}$:

$$\begin{aligned} X \in \mathsf{Con} &\iff X \cap A \in \mathsf{Con}_A \text{ und } X \cap B \in \mathsf{Con}_B, \\ X \vdash c &\iff (c \in A \Rightarrow X \cap A \vdash_A c) \text{ und } (c \in B \Rightarrow X \cap B \vdash_B c). \end{aligned}$$

Offenbar haben wir dann

LEMMA 4.1. *Seien $\boldsymbol{A}$ und $\boldsymbol{B}$ Informationssysteme mit $A \cap B = \varnothing$. Dann ist auch $\boldsymbol{A} \times \boldsymbol{B}$ ein Informationssystem. Die Objekte von $\boldsymbol{A} \times \boldsymbol{B}$ sind genau die Vereinigungen $x \cup y$ von Objekten $x \in |\boldsymbol{A}|$ und $y \in |\boldsymbol{B}|$.* □

Beim Aufbau von Informationssystemen mit Hilfe von $\to$ und $\times$ nehmen wir an, daß $\times$ stärker bindet als $\to$. Etwa $\boldsymbol{A} \times \boldsymbol{B} \to \boldsymbol{C}$ bedeutet also $(\boldsymbol{A} \times \boldsymbol{B}) \to \boldsymbol{C}$.

BEMERKUNG. Offenbar stehen die Paare $(x, y) \in |\boldsymbol{A}| \times |\boldsymbol{B}|$ in einer natürlichen Korrespondenz zu den Objekten $x \cup y$ von $|\boldsymbol{A} \times \boldsymbol{B}|$. Aus Gründen der Lesbarkeit schreiben wir deshalb meist x, y anstelle von $x \cup y$. Aus dem Kontext sollte es immer klar sein, wo ein Komma durch ein Vereinigungszeichen zu ersetzen ist.

Die *Projektionen* $\pi_i \colon \boldsymbol{A}_1 \times \boldsymbol{A}_2 \to \boldsymbol{A}_i$, definiert durch $\pi_i(x_1, x_2) := x_i$ für $i = 1, 2$, sind offenbar stetig.

LEMMA 4.2 (Universelle Eigenschaft kartesischer Produkte). *Seien $\boldsymbol{A}$, $\boldsymbol{B}$ und $\boldsymbol{C}$ Informationssysteme mit $A \cap B = \varnothing$. Für jedes Paar $f \colon \boldsymbol{C} \to \boldsymbol{A}$ und $g \colon \boldsymbol{C} \to \boldsymbol{B}$ stetiger Funktionen gibt es genau eine stetige Funktion $h \colon \boldsymbol{C} \to \boldsymbol{A} \times \boldsymbol{B}$ so daß $f = \pi_1 \circ h$ und $g = \pi_1 \circ h$.*

$$\begin{tikzcd} & \boldsymbol{A} \\ \boldsymbol{C} \arrow[ur, "f"] \arrow[r, "h"] \arrow[dr, "g"'] & \boldsymbol{A} \times \boldsymbol{B} \arrow[u, "\pi_1"'] \arrow[d, "\pi_2"] \\ & \boldsymbol{B} \end{tikzcd}$$

BEWEIS. Die Eindeutigkeit folgt daraus, daß für jedes solche h gelten muß $h(x) = (f(x), g(x))$ für $x \in |\boldsymbol{C}|$. Zum Beweis der Existenz definieren wir $h(x) := (f(x), g(x))$ für $x \in |\boldsymbol{C}|$. Zu zeigen ist, daß h stetig ist. Die Monotonie ist klar; für PFS nehmen wir $b \in (f(x), g(x))$ an. Wegen $A \cap B = \varnothing$ liegt das Datenobjekt b in genau einer Komponente, etwa B. Also $b \in g(x)$, daher $b \in g(\overline{X})$ für ein $X \subseteq^{\mathsf{fin}} x$ und deshalb $b \in h(\overline{X}) = (f(\overline{X}), g(\overline{X}))$. □

Als Anwendung konstruieren wir das *Produkt* $f \times g$ zweier stetiger Funktionen $f\colon \boldsymbol{A} \to \boldsymbol{C}$ und $g\colon \boldsymbol{B} \to \boldsymbol{D}$ (mit $A \cap B = \varnothing$):

$$\begin{tikzcd}
 & \boldsymbol{C} \\
\boldsymbol{A} \times \boldsymbol{B} \arrow[ur, "f \circ \pi_1"] \arrow[r, "f \times g"] \arrow[dr, "g \circ \pi_2"'] & \boldsymbol{C} \times \boldsymbol{D} \arrow[u, "\pi_1"'] \arrow[d, "\pi_2"] \\
 & \boldsymbol{D}
\end{tikzcd}$$

$f \times g$ erfüllt $(f \times g)(x, y) = (f(x), g(y))$ für $x \in |\boldsymbol{A}|$ und $y \in |\boldsymbol{B}|$.

LEMMA 4.3. *Seien $\boldsymbol{A}$, $\boldsymbol{B}$ und $\boldsymbol{C}$ Informationssysteme mit $A \cap B = \varnothing$. Eine Funktion $f\colon \boldsymbol{A} \times \boldsymbol{B} \to \boldsymbol{C}$ ist stetig genau dann, wenn sie in jedem Argument stetig ist, d.h. alle Schnitte $f_1^y\colon \boldsymbol{A} \to \boldsymbol{C}$, $f_1^y(x) := f(x, y)$ für festes $y \in |\boldsymbol{B}|$ und ebenso $f_2^x\colon \boldsymbol{B} \to \boldsymbol{C}$, $f_2^x(y) := f(x, y)$ für festes $x \in |\boldsymbol{A}|$ stetig sind.*

BEWEIS. Die Funktion, die für festes y jedem x das Paar (x, y) zuordnet, ist offenbar stetig, also (als Komposition zweier stetiger Funktionen) auch f_1^y; für f_2^x argumentiert man entsprechend. Seien umgekehrt alle f_1^y und f_2^x stetig. Zum Beweis der Monotonie von f nehmen wir an, daß $x \subseteq x'$ und $y \subseteq y'$ mit $x, x' \in |\boldsymbol{A}|$ und $y, y' \in |\boldsymbol{B}|$. Dann gilt $f(x, y) \subseteq f(x', y) \subseteq f(x', y')$ wegen der Monotonie der Schnitte. Zum Beweis von PFS für f sei $c \in f(x, y)$. Dann gilt $c \in f_1^x(y)$, also

$$c \in f_1^x(\overline{Y}) = f(x, \overline{Y}) = f_2^{\overline{Y}}(x)$$

für ein $Y \subseteq^{\mathsf{fin}} y$ nach PFS für f_1^x, und auch

$$c \in f_2^{\overline{Y}}(\overline{X}) = f(\overline{X}, \overline{Y}) = f(\overline{X \cup Y})$$

für ein $X \subseteq^{\mathsf{fin}} x$ nach PFS für $f_2^{\overline{Y}}$. □

LEMMA 4.4. *Seien $\boldsymbol{A}$ und $\boldsymbol{B}$ Informationssysteme. Dann ist die Funktion*

$$\begin{aligned}
&\mathsf{eval}\colon (\boldsymbol{A} \to \boldsymbol{B}) \times \boldsymbol{A} \to \boldsymbol{B} \\
&\mathsf{eval}(f, x) := f(x) \quad \textit{für } f \in |\boldsymbol{A} \to \boldsymbol{B}|,\ x \in |\boldsymbol{A}|
\end{aligned}$$

stetig.

BEWEIS. Aus Gründen der Klarheit schreiben wir r anstelle von f, und verwenden auch die korrekete Bezeichnungsweise $|r|(x)$ anstelle von $r(x)$.

Nach dem vorangehenden Lemma genügt es, die Stetigkeit in jeder Komponente separat zu beweisen. Für die zweite Komponente ist dies klar. Für die erste Komponente folgt die Monotonie aus der Definition von $|r|$. Für

PFS sei $b \in \mathsf{eval}(r, x) = |r|(x)$. Dann gilt $r(X, b)$ für ein $X \subseteq^{\mathsf{fin}} x$, also mit $W := \{(X, b)\}$ gilt $W \subseteq^{\mathsf{fin}} r$ und

$$\mathsf{eval}(\overline{W}, x) = |\overline{W}|(x) = \{\, c \in B \mid \overline{W}(Y, c) \text{ für ein } Y \subseteq^{\mathsf{fin}} x \,\} \ni b$$

nach Definition von W. □

LEMMA 4.5. *Seien $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ Informationssysteme mit $A \cap B = \varnothing$. Dann ist die Funktion*

$$\mathsf{curry}\colon (\boldsymbol{A} \times \boldsymbol{B} \to \boldsymbol{C}) \to (\boldsymbol{A} \to (\boldsymbol{B} \to \boldsymbol{C}))$$
$$\mathsf{curry}(f)(x)(y) := f(x, y) \quad \textit{für } f \in |\boldsymbol{A} \times \boldsymbol{B} \to \boldsymbol{C}|,\ x \in |\boldsymbol{A}| \textit{ und } y \in |\boldsymbol{B}|$$

wohldefiniert und stetig.

BEWEIS. Wir schreiben wieder r anstelle von f, und verwenden die korrekete Bezeichnungsweise $|r|(x)$ anstelle von $r(x)$.

Sei $r \in |\boldsymbol{A} \times \boldsymbol{B} \to \boldsymbol{C}|$. Dann ist für festes $x \in |\boldsymbol{A}|$ die Funktion $k := |r|_2^x$ stetig, da sie jedem $y \in |\boldsymbol{B}|$ das Element $|r|(x, y) \in |\boldsymbol{C}|$ zuordnet und deshalb ein Schnitt von $|r|$ ist. Wir zeigen jetzt, daß auch die Funktion h stetig ist, die jedem $x \in |\boldsymbol{A}|$ das Element $\hat{k} \in |\boldsymbol{B} \to \boldsymbol{C}|$ zuordnet. Für die Monotonie nehmen wir $x \subseteq x'$ an. Sei $\hat{k}' = h(x')$. Zu zeigen ist $\hat{k} \subseteq \hat{k}'$. Es genügt zu zeigen, daß aus $\hat{k}(Y, c)$ folgt $\hat{k}'(Y, c)$, d.h. daß aus $c \in k(\overline{Y})$ folgt $c \in k'(\overline{Y})$. Aber $k(\overline{Y}) \subseteq k'(\overline{Y})$ gilt wegen der Monotonie von $|r|$. Für PFS sei $(Y, c) \in h(x) = \hat{k}$. Gesucht ist ein $X \subseteq^{\mathsf{fin}} x$ so daß $(Y, c) \in h(\overline{X})$. Wegen $(Y, c) \in \hat{k}$ ist $c \in k(\overline{Y}) = |r|(x, \overline{Y})$. PFS für den Schnitt $|r|_1^{\overline{Y}}$ liefert ein $X \subseteq^{\mathsf{fin}} x$ mit $c \in |r|(\overline{X}, \overline{Y}) = |h(\overline{X})|(\overline{Y})$, d.h. $(Y, c) \in h(\overline{X})$.

Zu zeigen bleibt die Stetigkeit der Funktion g, die jedem r das Element $\hat{h}$ zuordnet. Die Monotonie folgt aus der Definition von $|r|$. Für PFS gelte $(X, (Y, c)) \in g(r)$, für ein $X \in \mathsf{Con}_A$, $Y \in \mathsf{Con}_B$ und $c \in C$. Man zeigt dann leicht $(X, (Y, c)) \in g(\overline{W})$ für $W := \{(X \cup Y, c)\} \subseteq^{\mathsf{fin}} r$. □

Die Aussage des Lemmas läßt sich auch wie folgt formulieren: Zu jedem stetigen $f\colon \boldsymbol{A} \times \boldsymbol{B} \to \boldsymbol{C}$ gibt es ein stetiges $c_f\colon \boldsymbol{A} \to (\boldsymbol{B} \to \boldsymbol{C})$ so daß $\mathsf{eval} \circ (c_f \times \mathsf{id}) = f$:

$$\begin{tikzcd}
(\boldsymbol{B} \to \boldsymbol{C}) \times \boldsymbol{B} \arrow[r, "\mathsf{eval}"] & \boldsymbol{C} \\
\boldsymbol{A} \times \boldsymbol{B} \arrow[u, "c_f \times \mathsf{id}"] \arrow[ur, "f"'] &
\end{tikzcd}$$

Ferner ist ein solches c_f offenbar eindeutig bestimmt. Dies zusammen mit der Tatsache, daß wir kartesische Produkte $\boldsymbol{A} \times \boldsymbol{B}$ bilden können (mit der universellen Eigenschaft aus Lemma 4.2) und der Existenz eines „terminalen" Informationssystems $(\varnothing, \{\varnothing\}, \varnothing)$ (derart daß es für jedes $\boldsymbol{A}$ ein eindeutiges $f\colon \boldsymbol{A} \to (\varnothing, \{\varnothing\}, \varnothing)$ gibt), beinhaltet, daß die Informationssysteme mit den stetigen Abbildungen zwischen ihnen eine *kartesisch abgeschlossene Kategorie* bilden.

Für das beabsichtigte Studium der Berechenbarkeit ist es wichtig, daß wir in dieser Kategorie auch kleinste Fixpunkte behandeln können. Dies wollen als nächstes tun.

Der wesentliche Begriff beim Studium der Berechenbarkeit ist die Rekursion, und Lösungen für rekursive Definitionsgleichungen lassen sich als kleinste Fixpunkte gewisser höherstufiger Operatoren konstruieren. Zum Beispiel kann man die Addition definieren durch

$$m + n = Y_{\mathbf{N}\to\mathbf{N}}(\lambda\varphi\lambda x.\ \text{if } x = 0 \text{ then } m \text{ else } \varphi(x-1)+1 \text{ fi})n,$$

wobei $Y_{\mathbf{N}\to\mathbf{N}}$ jedem Funktional vom Typ $(\mathbf{N} \to \mathbf{N}) \to (\mathbf{N} \to \mathbf{N})$ seinen kleinsten Fixpunkt vom Typ $\mathbf{N} \to \mathbf{N}$ zuordnet; wir werden gleich sehen, daß ein solches Y immer existiert. Dieser Ansatz läßt sich unmittelbar auf höhere Typen erweitern: man betrachtet dann Y_ρ vom Typ $(\rho \to \rho) \to \rho$.

Sei $\boldsymbol{A}$ ein Informationssystem und $f\colon \boldsymbol{A} \to \boldsymbol{A}$. Ein Objekt $x \in |\boldsymbol{A}|$ ist ein *Fixpunkt* von f, wenn $f(x) = x$. Es heißt kleinster Fixpunkt von f, wenn für jedes andere $y \in |\boldsymbol{A}|$ mit $f(y) = y$ gilt $x \subseteq y$.

Satz 4.6 (Fixpunktsatz). *Sei $\boldsymbol{A}$ ein Informationssystem und $f\colon \boldsymbol{A} \to \boldsymbol{A}$. Dann hat f einen kleinsten Fixpoint $Y(f)$, und zwar*

$$Y(f) := \bigcup_{n\in\mathbb{N}} f^n(\overline{\varnothing}).$$

Ferner ist Y stetig.

Beweis. Aus $\overline{\varnothing} \subseteq f(\overline{\varnothing})$ erhält man $f^n(\overline{\varnothing}) \subseteq f^{n+1}(\overline{\varnothing})$ durch Induktion, da f monoton ist. Also ist $x := \bigcup_n f^n(\overline{\varnothing}) \in |\boldsymbol{A}|$. Ferner ergibt sich aus der Charakterisierung der Stetigkeit in Lemma 3.7(c)

$$f(x) = f\big(\bigcup_n f^n(\overline{\varnothing})\big) = \bigcup_n f^{n+1}(\overline{\varnothing}) = x.$$

Ist y ein weiterer Fixpunkt, so folgt aus $\overline{\varnothing} \subseteq y$ wieder durch Induktion $f^n(\overline{\varnothing}) \subseteq y$, also

$$x = \bigcup_n f^n(\overline{\varnothing}) \subseteq y.$$

Schließlich zeigen wir noch, daß Y stetig ist. Die Monotonie folgt unmittelbar aus der Definition. Für PFS gelte $b \in Y(f)$, d.h. $b \in f^m(\overline{\varnothing})$ für ein m. Da nun die Auswertung stetig ist, ist für jedes feste m auch das Funktional $f \mapsto f^m(\overline{\varnothing})$ stetig. Also ergibt sich aus PFS für unser $b \in f^m(\overline{\varnothing})$ die Existenz eines $W \subseteq^{\mathsf{fin}} f$ mit

$$b \in \overline{W}^m(\overline{\varnothing}) \subseteq Y(\overline{W}).$$

Das war zu zeigen. □

5. Partielle stetige Funktionale

Eine wichtige Konsequenz des Arbeitens in einer kartesisch abgeschlossenen Kategorie ist es, daß jede *explizite Definition* (die auch höhere Typen enthalten darf) tatsächlich ein Objekt definiert.

Unter einer expliziten Definition verstehen wir einen reinen λ-Term, wie dies in Kapitel 3, Definition 1.2 erklärt war. Der Einfachheit halber nehmen

wir an, daß nur ein Grundtyp ι gegeben ist, etwa der Typ $\mathbf{N}$ der natürlichen Zahlen. Die *Stufe* $\mathsf{lev}(\rho)$ eines Typs ρ wird induktiv erklärt durch

$$\begin{aligned}\mathsf{lev}(\iota) &:= 0\\ \mathsf{lev}(\rho \to \sigma) &:= \max(\mathsf{lev}(\rho)+1, \mathsf{lev}(\sigma))\\ \mathsf{lev}(\rho \times \sigma) &:= \max(\mathsf{lev}(\rho), \mathsf{lev}(\sigma)).\end{aligned}$$

Wir verwenden die Klammerkonventionen aus Kapitel 3. Man beachte, daß jeder Typ ohne $\times$ sich eindeutig schreiben läßt in der Form $\rho_1 \to \cdots \to \rho_n \to \iota$, und daß seine Stufe gegeben ist durch $\max(\mathsf{lev}(\rho_1), \ldots, \mathsf{lev}(\rho_n)) + 1$.

DEFINITION 5.1. Für jeden Typ ρ definieren wir das Informationssystem $\boldsymbol{C}_\rho = (C_\rho, \mathsf{Con}_\rho, \vdash_\rho)$ durch Induktion über ρ:

$$\begin{aligned}\boldsymbol{C}_\iota &:= \mathbb{N} \quad \text{(etwa als flaches Informationssystem; s. Beispiel 2.4(a))}\\ \boldsymbol{C}_{\rho\to\sigma} &:= \boldsymbol{C}_\rho \to \boldsymbol{C}_\sigma\\ \boldsymbol{C}_{\rho\times\sigma} &:= \boldsymbol{C}_\rho \times \boldsymbol{C}_\sigma.\end{aligned}$$

Die *partiellen stetigen Funktionale* (über $\mathbb{N}$) vom Typ $\rho \to \sigma$ sind die stetigen Funktionen von $|\boldsymbol{C}_\rho|$ nach $|\boldsymbol{C}_\sigma|$.

LEMMA 5.2. *Für jeden λ-Term M^ρ und jede Liste $x_1^{\rho_1}, \ldots, x_n^{\rho_n}$, die alle in M freien Variablen enthält, können wir $[\![M]\!]\colon \boldsymbol{C}_{\rho_1} \times \cdots \times \boldsymbol{C}_{\rho_n} \to \boldsymbol{C}_\rho$ so definieren, daß für alle $u_1 \in |\boldsymbol{C}_{\rho_1}|, \ldots, u_n \in |\boldsymbol{C}_{\rho_n}|$ gilt*

$$\begin{aligned}[\![x_i]\!](u_1, \ldots, u_n) &= u_i\\ [\![MN]\!](u_1, \ldots, u_n) &= [\![M]\!](u_1, \ldots, u_n)([\![N]\!](u_1, \ldots, u_n))\\ [\![\lambda x M]\!](u_1, \ldots, u_n)(v) &= [\![M]\!](u_1, \ldots, u_n, v).\end{aligned}$$

BEWEIS. Durch Induktion über den Term M. Ist M eine Variable x_i, so sei $[\![x_i]\!] := \pi_i^n$, wobei π_i^n die Projektion eines n-Tupels auf seine i-te Komponente ist. Formal definieren wir π_i^n aus den Projektionen π_1, π_2 (siehe oben) wie folgt (man beachte, daß $A \times B \times C := (A \times B) \times C$): $\pi_1^1 = \mathsf{id}$, $\pi_{n+1}^{n+1} = \pi_2$, $\pi_i^{n+1} = \pi_i^n \circ \pi_1$ für $i = 1, \ldots, n$.

Für einen Anwendungsterm definieren wir $[\![MN]\!] := \mathsf{eval} \circ ([\![M]\!] \times [\![N]\!])$. Dies ist eine stetige Funktion, und es gilt

$$\begin{aligned}[\![MN]\!](u_1, \ldots, u_n) &= \mathsf{eval}\big(([\![M]\!] \times [\![N]\!])(u_1, \ldots, u_n)\big)\\ &= \mathsf{eval}([\![M]\!](u_1, \ldots, u_n), [\![N]\!](u_1, \ldots, u_n))\\ &= [\![M]\!](u_1, \ldots, u_n)([\![N]\!](u_1, \ldots, u_n)).\end{aligned}$$

Im Fall einer λ-Abstraktion setzen wir $[\![\lambda x M]\!] := \mathsf{curry}([\![M]\!])$. Dann gilt aufgrund der Eigenschaften von curry

$$[\![\lambda x M]\!](u_1, \ldots, u_n)(v) = \mathsf{curry}([\![M]\!])(u_1, \ldots, u_n)(v) = [\![M]\!](u_1, \ldots, u_n, v).$$

Das war zu zeigen. □

6. Berechenbare Funktionale

In unseren Funktionenräumen $\boldsymbol{C}_\rho$ können wir jetzt eine Theorie der Berechenbarkeit entwickeln. Die Datenobjekte und daraus gebildete endliche Mengen lassen sich etwa durch natürliche Zahlen kodieren. Es ist dann leicht zu sehen, daß die Begriffe $X \in \mathsf{Con}_\rho$ der Konsistenz und $X \vdash_\rho a$ der Folgerungsrelation berechenbare (sogar „elementare“) Relationen sind.

DEFINITION 6.1. Ein partielles stetiges Funktional φ vom Typ ρ heißt be *berechenbar* wenn es – angesehen als Menge von (Kodenummern von) Datenobjekten – Σ_1-definierbar ist.

LEMMA 6.2. *Für alle Typen ρ, σ, τ sind die Funktionale*

$$\mathsf{eval}_{\rho,\sigma}\colon (\boldsymbol{C}_\rho \to \boldsymbol{C}_\sigma) \times \boldsymbol{C}_\rho \to \boldsymbol{C}_\sigma$$
$$\mathsf{curry}_{\rho,\sigma,\tau}\colon (\boldsymbol{C}_\rho \times \boldsymbol{C}_\sigma \to \boldsymbol{C}_\tau) \to (\boldsymbol{C}_\rho \to (\boldsymbol{C}_\sigma \to \boldsymbol{C}_\tau))$$

berechenbar.

BEWEIS. Die Datenobjekte von $\mathsf{eval}_{\rho,\sigma}$ sind von der Form (W, X, a) mit $W \in \mathsf{Con}_{\rho\to\sigma}$, $X \in \mathsf{Con}_\rho$ und $a \in C_\sigma$, und es gilt

$$(W, X, a) \in \mathsf{eval} \iff a \in \overline{W}(\overline{X}) \iff a \in \overline{WX} \iff WX \vdash a.$$

Hierbei ist WX die in Definition 3.1 eingeführte Anwendungsoperation für endliche Approximationen, und wir haben die (einfach zu beweisende) Tatsache verwendet, daß $\overline{WX} = \overline{W}(\overline{X})$. Also haben wir eine Σ_1-Definition von $\mathsf{eval}_{\rho,\sigma}$. Für curry erhalten wir

$$\begin{aligned}(W, X, Y, a) \in \mathsf{curry} &\iff a \in \overline{W}(\overline{X \cup Y})\\ &\iff a \in \overline{W(X \cup Y)}\\ &\iff W(X \cup Y) \vdash a,\end{aligned}$$

was wieder eine Σ_1-Definition ist. □

LEMMA 6.3. (a) *Sind $\varphi\colon \boldsymbol{C}_\rho \to \boldsymbol{C}_\sigma$ und $\psi\colon \boldsymbol{C}_\sigma \to \boldsymbol{C}_\tau$ berechenbar, so auch $\psi \circ \varphi$.*
(b) *Sind $\Phi\colon \boldsymbol{C}_\rho \to \boldsymbol{C}_\sigma$ und $\varphi \in |\boldsymbol{C}_\rho|$ berechenbar, so auch $\Phi(\varphi)$.*
(c) *Sind $\varphi\colon \boldsymbol{C}_\rho \to \boldsymbol{C}_\sigma$ und $\psi\colon \boldsymbol{C}_\rho \to \boldsymbol{C}_\tau$ berechenbar, so auch $\varphi \times \psi$.*
(d) *Die Projektionen $\pi_1\colon \boldsymbol{C}_\rho \times \boldsymbol{C}_\sigma \to \boldsymbol{C}_\rho$ und $\pi_2\colon \boldsymbol{C}_\rho \times \boldsymbol{C}_\sigma \to \boldsymbol{C}_\sigma$ sind berechenbar.*

BEWEIS. (a). Die Datenobjekte von $\psi \circ \varphi$ sind von der Form (X, b) mit $X \in \mathsf{Con}_\rho$ und $b \in C_\tau$, und es gilt

$$(X, b) \in \psi \circ \varphi \iff \exists Y \subseteq^{\mathsf{fin}} \varphi[(Y, b) \in \psi \wedge \forall a \in Y\, (X, a) \in \varphi].$$

Dies ist eine Σ_1-Definition, da φ und ψ berechenbar sind.

(b). Da eval stetig ist, haben wir wegen PFS

$$\begin{aligned}a \in \Phi(\varphi) &\iff \exists W \subseteq^{\mathsf{fin}} \Phi \exists X \subseteq^{\mathsf{fin}} \varphi(a \in \overline{W}(\overline{X}))\\ &\iff \exists W \subseteq^{\mathsf{fin}} \Phi \exists X \subseteq^{\mathsf{fin}} \varphi(WX \vdash a).\end{aligned}$$

Dies ist wieder eine Σ_1-Definition.

(c) und (d) beweist man ähnlich. □

LEMMA 6.4. *Für jeden λ-Term M^ρ und jede Liste $x_1^{\rho_1}, \ldots, x_n^{\rho_n}$ von Variablen, die alle in M freien Variablen enthält, ist das wie oben definierte Funktional $[\![M]\!]\colon \boldsymbol{C}_{\rho_1} \times \cdots \times \boldsymbol{C}_{\rho_n} \to \boldsymbol{C}_\rho$ berechenbar.*

BEWEIS. Die Berechenbarkeit folgt aus der Konstruktion von $[\![M]\!]$ in Lemma 5.2 (durch Induktion über M) und dem vorangehenden Lemma. □

Man kann ferner leicht zeigen, daß die Fixpunktoperatoren Y_ρ berechenbar sind, sowie auch die Funktionale pcond vom Typ $\iota \to \iota \to \iota \to \iota$ und $\exists$ vom Typ $(\iota \to \iota) \to \iota$, die definiert sind durch

$$\mathsf{pcond}(p, x, y) := \begin{cases} x & \text{falls } p = 1 \\ y & \text{falls } p = 0 \\ x & \text{falls } x = y \\ \bot & \text{sonst,} \end{cases}$$

$$\exists(\varphi) := \begin{cases} 0 & \text{falls } \varphi(\bot) = 0 \\ 1 & \text{falls } \varphi(n) = 1 \text{ für ein } n \\ \bot & \text{sonst.} \end{cases}$$

DEFINITION 6.5. Ein partielles stetiges Funktional Φ vom Typ $\rho_1 \to \cdots \to \rho_p \to \iota$ heißt *rekursiv in* pcond *und* $\exists$ wenn es explizit durch einen λ-Term definierbar ist, der aus Variablen und Konstanten für 0, den Nachfolger, den Vorgänger, die Fixpunktoperatoren Y_ρ, pcond und $\exists$ aufgebaut ist.

SATZ 6.6 (PLOTKIN). *Ein partielles stetiges Funktional ist berechenbar genau dann, wenn es rekursiv in* pcond *und* $\exists$ *ist.*

BEWEIS. Die obigen Beispiele zusammen mit den Tatsachen, daß (1) λ-Terme berechenbar sind (siehe Lemma 6.4) und (2) berechenbare Funktionale abgeschlossenen sind unter Anwendungen (siehe Lemma 6.3(a)) ergibt sich, daß jedes in pcond and $\exists$ rekursive Funktional berechenbar ist. Für die Umkehrung sei auf die Literatur [**6**] verwiesen. □

ANHANG A

MPC-Beweis zum Warshall-Algorithmus

Als ein Beispiel für eine nicht mehr ganz einfache logische Herleitung wollen wir jetzt den in Kapitel 4, Abschnitt 4 informal geführten Beweis für die dem Warshall-Algorithmus entsprechende Existenzaussage formalisieren. Das ist notwendig, wenn man anschließend ein Programm automatisch extrahieren möchte.

Wir verwenden den Beweisprüfer MPC zur formalen Darstellung des Beweises. Erreicht wird damit, daß der Beweis maschinell auf Korrektheit überprüft ist.

Wir stellen zunächst Dateien `nat.scm` und `list.scm` bereit; dies sind Bibliotheksdateien, die Grundmaterial für natürliche Zahlen und Listen enthalten, einschließlich den Namen n, m, k für Zahlvariablen, den Bezeichnungen $++i$ für den Nachfolger und der Infixbezeichnung :: für `cons`. Ferner legen wir weitere Namen für Variablen fest, die über natürliche Zahlen bzw. Listen von natürlichen Zahlen laufen, und auch einen Namen r für zweistellige booleschwertige Funktionen über natürlichen Zahlen:

```
MPC;
INCLUDE "nat.mpc";
INCLUDE "list.mpc";

nat .                    a b c d e i j;
(list nat) .             x y z;
nat=>nat=>boole .        r; // the relation, mostly supressed

SYNTAX nil CONST (Nil nat);
```

In Kapitel 4, Abschnitt 4 haben wir die folgenden Prädikaten- und Funktionssymbole verwendet. Zugrunde lag eine zweistellige Relation R auf $\{0, 1, \ldots, n-1\}$, deren transitive Hülle zu bestimmen war.

$k \in x$	k kommt im Pfad x vor,
$\mathsf{Rf}(x)$	x ist ein wiederholungsfreier Pfad,
$P_i(x, j, k)$	x ist ein R-Pfad von j nach k mit inneren Elementen $< i$,
$x \mid y$	Verkettung.

Die Verkettungsfunktion $x \mid y$ leistete folgendes. Sind x und y Pfade mit gleichem End- bzw. Anfangspunkt, so ist $x \mid y$ der durch Aneinanderhängen entstehende Pfad, wobei der gleiche End- und Anfangspunkt nur einmal aufgeführt wird. Sind End- und Anfangspunkt verschieden, so ist das Ergebnis die leere Liste ϵ.

Wir haben damals keine Definitionen dieser Relationen und Funktionen angegeben; dies müssen wir jetzt nachholen. Aus den oben genannten Gründen schreiben wir die Definitionsgleichungen als *Termersetzungsregeln*.

```
FUNCTION boole . In(nat (list nat))
  // k in x: k occurs in the path x
{ SYNTAX in RELOP In;
  a in nil -> False;
  a in (b::x) -> [if (a=b) True (a in x)];
};

FUNCTION boole . Rf(list nat)
  // Rf x : x is a repetition free path
{ Rf nil -> True;
  Rf(a::x) -> [if (a in x) False (Rf x)];
};

FUNCTION boole . Path(nat=>nat=>boole nat (list nat) nat nat)
{  SYNTAX P CONST (Path r);
  // we use P and suppress the notation of r
   P i nil b c -> False;
   P i (a::nil) b c -> [if (a=b) (b=c) False];
   P i (a::d::nil) b c ->
         [if (a=b) [if (d=c) (r a d) False]  False];
   P i (a::d::e::x) b c ->
         [if (a=b)
             [if (r a d)
                 [if (d<i) (P i (d::e::x) d c) False]
                 False]
             False];
 };

FUNCTION list nat . Dot((list nat) (list nat))
{SYNTAX | PAIROP Dot;
 nil | x -> x;
 (a::nil) | nil -> a:;
 (a::nil) | (b::x) -> [if (a=b) (b::x) nil];
 (a::d::x) | y -> a::(d::x | y);
};
```

Wir hatten einige Eigenschaften („Lemmata“) ohne „rechnerischen Gehalt“ zugrunde gelegt. Sie werden in MPC als Annahmen aufgelistet:

```
all r,i,x,j,k . P i x j k -> P ++i x j k. // 0
all r,x,j,k . P 0 x j k -> not j=k -> r j k. // 1
all r,i,x,y,j,k . P i x j i -> P i y i k ->
  P ++i (x|y) j k. // 2
all r,i,x,y,j,k . P i x j i ->  Rf x -> P i y i k -> Rf y ->
  all z not P i z j k -> Rf(x|y). // 3

all r,i,x,j,k . P ++i x j k -> not P i x j k ->
```

```
(ex y P i y j i & ex z P i z i k). // 4
```

Ferner benötigen wir im formalen Beweis noch zwei weitere Aussagen, die sich ebenfalls und sogar besonders leicht beweisen lassen, deren Beweis wir der Kürze halber aber übergehen möchten.

```
all j,k . (not j=k) -> Rf(j::k::nil). // 5
all r,i,x,j,k . P i x j k -> not x=nil. // 6
```

Diese Annahmen werden von MPC mit den Nummern 0–6 versehen und im folgenden Beweis verwendet.

```
{ r.
// induction on i
//
// base i = 0
//
{ j. { k. // given j and k
  // cases j=k or not j=k
  { j=k.
    // let x be j::nil

    { (j::nil)=nil. { y . { P 0 y j k . F; }}};
    { not (j::nil)=nil.
      P 0 (j::nil) j k;  // from j=k we have P 0 (j::nil) j k;
      Rf(j::nil); // trivial
      (P 0 (j::nil) j k & Rf(j::nil));
    }
    ((j::nil)=nil -> all y not P 0 y j k) &
     (not (j::nil)=nil -> (P 0 (j::nil) j k & Rf(j::nil)));

    ex x . (x=nil -> all y not P 0 y j k) &
           (not x=nil -> (P 0 x j k & Rf x));
  }
  { not j=k.
    // cases r j k or not r j k
    { r j k.
      // let x be j::k::nil
     { (j::k::nil)=nil. { y. { P 0 y j k. F;}}}
     { not(j::k::nil)=nil.
       P 0 (j::k::nil) j k; // since r j k.
       // we use proposition 5
       all k . (not j=k) -> Rf(j::k::nil);
       (not j=k) -> Rf(j::k::nil);
       Rf(j::k::nil);
       P 0 (j::k::nil) j k & Rf(j::k::nil);
     }
     ((j::k::nil)=nil -> all y not P 0 y j k) &
      (not (j::k::nil)=nil -> (P 0 (j::k::nil) j k &
                               Rf(j::k::nil)));
      ex x . (x=nil -> all y not P 0 y j k) &
             (not x=nil -> (P 0 x j k & Rf x));
```

```
    }
    { not r j k.
      // let x be nil.
      { nil=nil. { y. { P 0 y j k.
        // we use proposition 1
        all x,j,k . P 0 x j k -> not j=k -> r j k;
        all j,k . P 0 y j k -> not j=k -> r j k;
        all k . P 0 y j k -> not j=k -> r j k;
        P 0 y j k -> not j=k -> r j k;
        not j=k -> r j k;
        r j k;
        F;
      }}}
      { not nil=nil.
        nil=nil;
        P 0 nil j k;
        Rf nil;
        P 0 nil j k & Rf nil;
      }

      (nil=nil -> all y not P 0 y j k) &
      (not nil=nil -> (P 0 nil j k & Rf nil));

      ex x . (x=nil -> all y not P 0 y j k) &
             (not x=nil -> (P 0 x j k & Rf x));
    }    ex x . (x=nil -> all y not P 0 y j k) &
                (not x=nil -> (P 0 x j k & Rf x));
  }
  ex x . (x=nil -> all y not P 0 y j k) &
         (not x=nil -> (P 0 x j k & Rf x));
}}
//
// step i -> ++i
//
{ i. { all j,k ex x . (x=nil -> all y not P i y j k) &
                      (not x=nil -> (P i x j k & Rf x)).

  { j. { k.
    // by induction hypothesis we have x1
    all k ex x . (x=nil -> all y not P i y j k) &
                 (not x=nil -> (P i x j k & Rf x));
    ex x . (x=nil -> all y not P i y j k) &
           (not x=nil -> (P i x j k & Rf x));
    { x1. { (x1=nil -> all y not P i y j k) &
            (not x1=nil -> (P i x1 j k & Rf x1)). // IH for x1
      // cases x1=nil or not x1=nil
      { x1=nil.
        x1=nil -> all y not P i y j k;
```

```
all y not P i y j k; // by IH
// now we get by IH a path x2 from j to i

all k ex x . (x=nil -> all y not P i y j k) &
             (not x=nil -> (P i x j k & Rf x));
ex x . (x=nil -> all y not P i y j i) &
       (not x=nil -> (P i x j i & Rf x));

{ x2. { (x2=nil -> all y not P i y j i) &
        (not x2=nil -> (P i x2 j i & Rf x2)).
  // cases x2=nil or not x2=nil
  { x2=nil.
    x2=nil -> all y not P i y j i;
    // then we have
    all y not P i y j i;
    // let x be nil
    { nil=nil. { y. { P ++i y j k.
      // we use proposition 4
      all i,x,j,k . P ++i x j k -> not P i x j k ->
          (ex y P i y j i & ex z P i z i k);
      all x,j,k . P ++i x j k -> not P i x j k ->
          (ex y P i y j i & ex z P i z i k);
      all j,k . P ++i y j k -> not P i y j k ->
          (ex z1 P i z1 j i & ex z2 P i z2 i k);
      all k . P ++i y j k -> not P i y j k ->
          (ex z1 P i z1 j i & ex z2 P i z2 i k);
      P ++i y j k -> not P i y j k ->
          (ex z1 P i z1 j i & ex z2 P i z2 i k);
      not P i y j k ->
          (ex z1 P i z1 j i & ex z2 P i z2 i k);
      not P i y j k;
        ex z1 P i z1 j i & ex z2 P i z2 i k;
      ex z1 P i z1 j i; // hence a contradiction
      { z1. { P i z1 j i.
        not P i z1 j i;
        F;
      }}
        F;
    }}}
    { not nil=nil.
      nil=nil;
      P ++i nil j k;
      Rf nil;
      P ++i nil j k & Rf nil;
    }
    (nil=nil -> all y not P ++i y j k) &
    (not nil=nil -> P ++i nil j k & Rf nil);
    ex x . (x=nil -> all y not P ++i y j k) &
```

```
          (not x=nil -> (P ++i x j k & Rf x));
}
{ not x2=nil.
   // now we get by IH a path x3 from i to k

  all k ex x . (x=nil -> all y not P i y i k) &
               (not x=nil -> (P i x i k & Rf x));
  ex x . (x=nil -> all y not P i y i k) &
               (not x=nil -> (P i x i k & Rf x));
  { x3. { (x3=nil -> all y not P i y i k) &
          (not x3=nil -> (P i x3 i k & Rf x3)).
    // cases x3=nil or not x3=nil
    { x3=nil.
      x3=nil -> all y not P i y i k;
      all y not P i y i k;
      // let x be nil
      { nil=nil. { y. { P ++i y j k.
        // we again use proposition 4
        all i,x,j,k. P ++i x j k -> not P i x j k ->
           (ex y P i y j i & ex z P i z i k);
        all x,j,k . P ++i x j k -> not P i x j k ->
           (ex y P i y j i & ex z P i z i k);
        all j,k . P ++i y j k -> not P i y j k ->
           (ex z1 P i z1 j i & ex z2 P i z2 i k);
        all k . P ++i y j k -> not P i y j k ->
           (ex z1 P i z1 j i & ex z2 P i z2 i k);
        P ++i y j k -> not P i y j k ->
           (ex z1 P i z1 j i & ex z2 P i z2 i k);
        not P i y j k ->
           (ex z1 P i z1 j i & ex z2 P i z2 i k);
        not P i y j k;
        ex z1 P i z1 j i & ex z2 P i z2 i k;
        ex z2 P i z2 i k; // and again contradiction
        { z2. { P i z2 i k.
          not P i z2 i k;
             F;
        }}
        F;
      }}}
      { not nil=nil.
        nil=nil;
        P ++i nil j k;
        Rf nil;
        (P ++i nil j k & Rf nil);
      }
      (nil=nil -> all y not P ++i y j k) &
      (not nil=nil -> (P ++i nil j k & Rf nil));
      ex x . (x=nil -> all y not P ++i y j k) &
```

```
        (not x=nil -> (P ++i x j k & Rf x));
 }
 { not x3=nil.
   // now let x be x2|x3.
   // We first prove not (x2 | x3)=nil
   // about x2 and x3 we know
   not x2=nil -> (P i x2 j i & Rf x2);
   P i x2 j i & Rf x2;
   P i x2 j i; Rf x2;

   not x3=nil -> (P i x3 i k & Rf x3);
   P i x3 i k & Rf x3;
   P i x3 i k; Rf x3;

   // we use proposition 2
   all i,x,y,j,k . P i x j i -> P i y i k ->
                   P ++i (x|y) j k;
   all x,y,j,k . P i x j i -> P i y i k ->
                 P ++i (x|y) j k;
   all y,j,k . P i x2 j i -> P i y i k ->
               P ++i (x2|y) j k;
   all j,k . P i x2 j i -> P i x3 i k ->
             P ++i (x2|x3) j k;
   all k . P i x2 j i -> P i x3 i k ->
           P ++i (x2|x3) j k;
   P i x2 j i -> P i x3 i k -> P ++i (x2|x3) j k;
   P i x3 i k -> P ++i (x2|x3) j k;
   P ++i (x2|x3) j k;

   // we use proposition 6
   all i,x,j,k . P i x j k -> not x=nil;
   all x,j,k . P ++i x j k -> not x=nil;
   all j,k . P ++i (x2|x3) j k -> not(x2|x3)=nil;
   all k . P ++i (x2|x3) j k -> not (x2|x3)=nil;
   P ++i (x2|x3) j k -> not (x2|x3)=nil;
   not (x2|x3)=nil;

   { (x2|x3)=nil. { y. { P ++i y j k.
     F;
   }}}

   { not (x2|x3)=nil.

     // we use proposition 3
     all i,x,y,j,k . P i x j i ->  Rf x ->
                     P i y i k -> Rf y ->
         all z not P i z j k -> Rf(x|y);
     all x,y,j,k . P i x j i ->  Rf x ->
```

```
                        P i y i k -> Rf y ->
                 all z not P i z j k -> Rf(x|y);
            all y,j,k . P i x2 j i ->  Rf x2 ->
                       P i y i k -> Rf y ->
                 all z not P i z j k -> Rf(x2|y);
            all j,k . P i x2 j i ->  Rf x2 ->
                      P i x3 i k -> Rf x3 ->
                 all z not P i z j k -> Rf(x2|x3);
            all k . P i x2 j i ->  Rf x2 ->
                     P i x3 i k -> Rf x3 ->
                 all z not P i z j k -> Rf(x2|x3);
            P i x2 j i ->  Rf x2 ->
               P i x3 i k -> Rf x3 ->
               all z not P i z j k -> Rf(x2|x3);
            Rf x2 -> P i x3 i k -> Rf x3 ->
               all z not P i z j k -> Rf(x2|x3);
            P i x3 i k -> Rf x3 ->
               all z not P i z j k -> Rf(x2|x3);
            Rf x3 ->
               all z not P i z j k -> Rf(x2|x3);
            all z not P i z j k -> Rf(x2|x3);
            all z not P i z j k;
            Rf(x2|x3);

            P ++i (x2|x3) j k & Rf(x2|x3);
          }
          ((x2|x3)=nil -> all y not P ++i y j k) &
          (not (x2|x3)=nil -> (P ++i (x2|x3) j k &
                               Rf(x2|x3)));
          ex x . (x=nil -> all y not P ++i y j k) &
                 (not x=nil -> (P ++i x j k & Rf x));
        }
        ex x . (x=nil -> all y not P ++i y j k) &
               (not x=nil -> (P ++i x j k & Rf x));
      }}
      ex x . (x=nil -> all y not P ++i y j k) &
             (not x=nil -> (P ++i x j k & Rf x));
    }
  ex x . (x=nil -> all y not P ++i y j k) &
         (not x=nil -> (P ++i x j k & Rf x));
  }}

  ex x . (x=nil -> all y not P ++i y j k) &
         (not x=nil -> (P ++i x j k & Rf x));
}
{ not x1=nil.
  // let x be x1
  not x1=nil -> (P i x1 j k & Rf x1);
```

```
          P i x1 j k & Rf x1;
          Rf x1;
          P i x1 j k;
          // we use proposition 0 to conclude P ++i x1 j k.
          all i,x,j,k. P i x j k -> P ++i x j k;
          all x,j,k. P i x j k -> P ++i x j k;
          all j,k. P i x1 j k -> P ++i x1 j k;
          all k. P i x1 j k -> P ++i x1 j k;
          P i x1 j k -> P ++i x1 j k;
          P ++i x1 j k;
          P ++i x1 j k & Rf x1;
          {not x1=nil. (P ++i x1 j k & Rf x1);}
          { x1=nil. { y. { P ++i y j k. F;}}}
          (x1=nil -> all y not P ++i y j k) &
          (not x1=nil -> (P ++i x1 j k & Rf x1));

          ex x . (x=nil -> all y not P ++i y j k) &
                 (not x=nil -> (P ++i x j k & Rf x));
        }
        ex x . (x=nil -> all y not P ++i y j k) &
               (not x=nil -> (P ++i x j k & Rf x));
    }}
    ex x . (x=nil -> all y not P ++i y j k) &
           (not x=nil -> (P ++i x j k & Rf x));
  }}
}}
// final formula
 all i,j,k ex x . (x=nil -> all y not P i y j k) &
                  (not x=nil -> (P i x j k & Rf x));
}
END;
```

Literaturverzeichnis

1. Reinhard Diestel, *Graphentheorie*, 2. ed., Springer Verlag, Berlin, Heidelberg, New York, 2000.
2. Gerhard Gentzen, *Untersuchungen über das logische Schließen*, Mathematische Zeitschrift **39** (1934), 176–210, 405–431.
3. Thomas Ihringer, *Diskrete Mathematik*, Teubner, Stuttgart, 1994.
4. Ingebrigt Johansson, *Der Minimalkalkül, ein reduzierter intuitionistischer Formalismus*, Compositio Mathematica **4** (1937), 119–136.
5. Kim G. Larsen and Glynn Winskel, *Using information systems to solve recursive domain equations*, Information and Computation **91** (1991), 232–258.
6. Gordon D. Plotkin, *LCF considered as a programming language*, Theoretical Computer Science **5** (1977), 223–255.
7. Kenneth H. Rosen, *Discrete mathematics and its applications*, Random House, New York, 1988.
8. Dana Scott, *Domains for denotational semantics*, Automata, Languages and Programming (E. Nielsen and E.M. Schmidt, eds.), LNCS, vol. 140, Springer Verlag, Berlin, Heidelberg, New York, 1982, A corrected and expanded version of a paper prepared for ICALP'82, Aarhus, Denmark, pp. 577–613.

Index

R-abgeschlossene Menge, 90
Äquivalenzrelation, 51

$\mathsf{dp}(\Gamma)$, 82
$|\Gamma|$, 82
Ableitung, 3
Abschlußzahl, 89
Abstraktion, 35
Addition, 38, 41
Adjazenzliste, 71
Adjazenzmatrix, 61
Adjazenzmenge, 71
Alexandrov-Bedingung, 101
Algebra
  finitäre, 35, 94
algebraisch, 97
Allformel, 13
Annahme, 3
  geschlossene, 3
  offene, 3
Annahmenvariablen, 3
Anwendung, 35, 98
append, 38
approximierbare Abbildung, 100
Argumenttyp, 33
Aristoteles, 2
Aussage, 2
Aussagensymbol, 14
Austauschlemma
  aus der linearen Algebra, 84
  der Graphentheorie, 84

Baum, 81
  aufspannender, 83
Berechenbarkeit, 94
Berechnungsregeln, 40
Bereich, 98
Beseitigungsregel, 1, 3, 4, 16
Beweis, 3
Bewertungsfunktion, 72
Blätterzahl, 82
Blatt
  eines Wurzelbaums, 82
Block, 5, 17
boole, 34
Brücke, 67

Cantorsches Diagonalverfahren, 51
complete partial ordering, 97
Cons, 34
cpo, 97
Curry-Howard Korrespondenz, 36

Datenobjekt, 95
De-Morgan-Äquivalenzen, 11
deduktiv abgeschlossen, 96
deduktive Abschluß, 97
Definition
  rekursive, 36
Disjunktion
  klassische, 24
  konstruktiver, 24
  schwache, 24
  starke, 24
Dummy, 34

Ecke, 61
Einführungsregel, 1, 3, 4, 16
Einselement, 52, 53
Elternteil, 82
erreichbarer Teil, 87
Ex-Falso-Quodlibet, 6, 19
Existenzformel, 13
Existenzquantor
  klassischer, 24
  konstruktiver, 24
  schwacher, 24
  starker, 24
explizite Definition, 107

Fallunterscheidung, 39, 43
Cases-Axiom, 43
Cases-Operator, 39
falsum, 2, 6, 14
Fixpunkt, 107
Folgerungsrelation, 95
Formel, 15, 43
  Σ_1, 93
  atomare, 15
  aussagenlogisch unzerlegbare, 22
  beschränkte, 93
  geschlossene, 15
  negative, 8, 20
  pränexe, 23
Formeln
  äquivalente, 7

freie Algebra, 33
Funktion
  berechenbare, 93
Funktional
  berechenbar, 109
  rekursiv in pcond und $\exists$, 110
Funktionssymbol, 14
Funktionstyp, 33

Gödel-Gentzen Übersetzung ^{g}, 8, 20
genau dann wenn, 2
gerichtet, 90
Geschwister, 82
Gleichheit, 42, 44
Größe eines Graphen, 82
Grad, 65
Graph, 61
  azyklischer, 81
  bewerteter, 71
  bipartiter, 64
  vollständig bipartiter, 64
  vollständiger, 64
  zusammenhängender, 63
Graphen
  isomorphe, 63
Grundterm, 15
Grundtyp, 33
Gruppentheorie, 13

Höhe eines Wurzelbaums, 82
Hülle
  reflexive, 51
  symmetrische, 51
  transitive, 51
Halbring, 53
Hauptprämisse, 4
herleitbar, 4, 25
Herleitung, 3

Ideal, 96
`if`-Konstrukt, 40
if-then-else, 40
Implikation, 2
`Ind`, 43
indirekter Beweis, 1
Induktionsprinzip, 88
induktive Definition, 87
Infix, 15
Informationssystem, 95
  flaches, 96
  Objekt eines, 96
Inl, 35, 39
Inr, 35, 39
Inzidenzmatrix, 62
isolierter Punkt, 63

Königsberger Brückenproblem, 66
Kante, 61
  hängende, 68
Kantengraph, 70
Kantenzug, 62
  geschlossener, 62
kartesisch abgeschlossene Kategorie, 106
Kern, 23
Kind, 82
Klammerkonventionen, 2, 15
Klauselform, 12
K_n, 64
$K_{n,m}$, 65
Knoten, 61
  äquivalente, 63
  hängender, 68
  innerer, eines Wurzelbaums, 82
kompakt, 97
Komprehensionsterm, 42
Konjunktion, 2
Konklusion, 3
konsistent, 96
konsistente Menge, 95
Konstante, 14
Konstruktor, 33
Konstruktormuster, 41
Konstruktortyp, 33
Konversionsrelation, 37
Kruskal-Algorithmus, 84

Länge, 72
Länge eines Graphen, 82
Logik
  intuitionistische, 6
  klassische, 6
  minimal, 25
  minimale, 4

Marke, 3
Matrizenprodukt, 53
minimal aufspannender Baum, 83
Minlog Proof Checker, 5
modus ponens, 4
monoton, 101
Moore-Algorithmus, 71
MPC, 5, 18, 25
Multiplikation, 38, 41

Nebenprämisse, 4
Nil, 34
Normalform
  disjunktive, 12, 22
  konjunktive, 12, 22

Objektvariablen, 17
oder, 2
Operator
  monotoner, 87
Ordnung
  partielle, eines Wurzelbaums, 82

Paarbildung, 35
Paartyp, 33
Parametertyp, 34
partielles stetiges Funktional, 108
Peirce-Formel, 7
PFS, 101
Prädikatenkonstante, 42
Prädikatensymbol, 14, 42
Prädikatenvariable, 42
Präfix, 23
Prämisse, 3
Pränexe Normalform, 23
Primformel, 15, 42
Principle of Finite Support, 101
Prinzip des indirekten Beweisens, 6
Produkt
  von Funktionen, 105
Programmkonstante, 40
Projektion, 104
Projektionen, 35

Quantor, 13
Quantorentiefe, 21

R-Pfad, 56
Reduktionssystem, 87
Regel, 4
  abgeleitete, 9, 21
  links-lineare, 41
reiner λ-Term, 35
Rekursionssatz, 92
Relation, 51
  eindeutige, 91
  fundierte, 87
  noethersche, 87
  terminierende, 87
Relationssymbol, 14
Ring, 52

Satz, 15
Scott Topologie, 101
Scott-Bedingung, 101
Scott-Ersov-Bereich, 98
Signatur, 14
spezielle Form, 40
Sprache
  erster Stufe, 14
Stabilität, 6, 19
Stelligkeit, 40
  einer Prädikatenvariablen, 42
stetig, 90
Strom, 96
Stufe, 108
Substitution, 16

Term, 14, 35
  geschlossener, 15, 35
Tertium non datur, 11
Tiefe, 82
Tiefe eines Wurzelbaums, 82
Typ, 33
  induktiv erzeugter, 33
Typvariable, 33

Umbenennung, 13
umdrehen
  von Listen, 38
und, 2

Variable, 13, 14
  Annahmen-, 3
  freie, 13, 15, 35
  gebundene, 13
Variablenbedingung, 17, 21
Verknüpfung, 51
vollständige partielle Ordnung, 97
Vorgängerfunktion, 40

Wahrheitsaxiom, 43
Wald, 81
  aufspannender, 83
Warshall-Algorithmus
  varallgemeinerter, 72
Weg, 62
  einfacher, 62
  Eulerscher, 65
  Hamiltonscher, 70
wenn–so, 2
Wurzel, 82
Wurzelbaum, 82

Zusammenhangskomponente, 63
Zykel, 62
Zyklus, 62
  Eulerscher, 65
  Hamiltonscher, 70