Ausdrucksmächtigkeit von Quantoren in algebraischen Spezifikationen

David Kempe

Karlsruhe, den 7. August 1998

Diplomarbeit

Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Logik, Komplexität und Deduktionssysteme

Betreuer:
Prof. Dr. Wolfram Menzel
Dr. Arno Schönegge

Hiermit versichere ich, die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Karlsruhe, den 7. August 1998

Inhaltsverzeichnis

Kapitel 1

Einleitung

In der Informatik sind abstrakte Datentypen allgegenwärtig. Seien es Listen, Bäume, Graphen oder Hashtabellen: fast alle Programme enthalten Algorithmen, die auf komplexen Datenstrukturen operieren, und deren Funktionalität kann einfach durch solche abstrakte Datentypen beschrieben werden.

Will man bei der Entwicklung zuverlässiger Programme eine möglichst hohe Sicherheit erreichen, kommt man kaum umhin, formale Spezifikationstechniken anzuwenden. Diese ermöglichen einerseits eine rigorose Verifikation des Verhaltens in Bezug auf die Implementierung, andererseits eine auf die Spezifikation zugeschnittene Generierung von Testdaten. Zur Spezifikation und Verifikation von Programmen gibt es verschiedene Werkzeuge — etwa das an der Universität Karlsruhe entwickelte KIV (Karlsruhe Interactive Verifier, vgl. [HMRS90, Rei95]).

Üblicherweise werden bei der algebraischen Spezifikation die gewünschten Eigenschaften des Datentyps mit Hilfe von Axiomen festgelegt. Je nach Art der Anwendung will man unter Umständen Eigenschaften des Datentyps unspezifiziert lassen, um Raum für Implementierungsfreiheiten zu gewähren, oder weil die Eigenschaften für die praktische Anwendung irrelevant sind. Dies ist etwa bei partiellen Funktionen der Fall — anstatt einen speziellen Funktionswert „undefiniert“ einzuführen, kann der Rückgabewert an undefinierten Stellen einfach unspezifiziert bleiben (vgl. etwa [GS95]).

Andererseits möchte man häufig auch über Datentypen argumentieren, bei denen alle Sorten und Funktionen (bis auf Isomorphie) eindeutig bestimmt sind. Solche Datentypen heißen monomorph.[1]

Die Sprache, in der die Axiome einer Spezifikation formuliert werden können, ist zumeist die Prädikatenlogik erster Stufe (kurz: PL) oder eine Teilmenge derselben. Allerdings möchte man für praktische Anwendungen häufig

[1]Exakte Definitionen der Begriffe werden in Kapitel 2 gegeben.

auch die Termerzeugtheit eines Datentyps fordern, d.h. die Eigenschaft, daß jedes Trägerelement des Datentyps durch einen variablenfreien Term dargestellt werden kann. Da diese Eigenschaft mittels Prädikatenlogik erster Stufe nicht axiomatisierbar ist, muß der Sprachschatz gegebenenfalls um entsprechende Konstrukte erweitert werden.

Eine weitere Eigenschaft, die die meisten in der Praxis auftretenden Datentypen aufweisen, ist die Berechenbarkeit. Denn wenn die Spezifikation der Verifikation oder Validierung einer Implementierung dienen soll, setzt dies ja insbesondere die Möglichkeit voraus, die Funktionen überhaupt zu implementieren. Es ist daher angebracht, wenn wir unsere Betrachtungen auf berechenbare Datentypen einschränken.

Wir wollen in dieser Arbeit zwei der wichtigsten Vertreter unter den für die Spezifikation verwendeten Sprachen untersuchen, und zwar einerseits die volle Prädikatenlogik erster Stufe und andererseits quantorenfreie Prädikatenlogik, beide erweitert um ein Konstrukt zur Spezifikation der Termerzeugtheit.

Bewußt verzichten wollen wir hingegen auf die Möglichkeit, die Signatur zum Zwecke der Spezifikation zu erweitern, d.h. Hilfsfunktionen hinzuzufügen, die für den „Benutzer" des Datentyps unsichtbar sind (und daher im Englischen „hidden functions" heißen). Solche Techniken können einerseits in Spezifikationssprachen und -werkzeugen zu technischen Problemen führen, andererseits werden die Spezifikationen damit häufig unnatürlich bzw. beginnen, funktionalen Programmen zu ähneln. Man läuft Gefahr, in der Spezifikation sozusagen schon die Programmierung umzusetzen.

Wir befassen uns also mit der Spezifikation berechenbarer, monomorpher, termerzeugter Datentypen ohne Verwendung von Hilfsfunktionen, mit Hilfe von PL auf der einen Seite und quantorenfreier Prädikatenlogik auf der anderen. Offensichtlich ist volle Prädikatenlogik mindestens so ausdrucksmächtig wie quantorenfreie, d.h. alle quantorenfrei spezifizierbaren Datentypen sind auch mit voller Prädikatenlogik spezifizierbar. Diese Arbeit wird der Antwort auf die Frage gewidmet sein, ob volle Prädikatenlogik echt ausdrucksstärker ist.

Während eine positive Antwort auf diese Frage naheliegend ist, besagt ein Resultat von Bergstra und Tucker ([BT82], [BT87]), daß unter Verwendung von höchstens $2n+3$ Hilfsfunktionen (wobei n die Anzahl der Sorten des Datentyps ist) jeder berechenbare, termerzeugte Datentyp quantorenfrei spezifizierbar ist. Wenn Hilfsfunktionen zugelassen sind, gewinnt man also durch Verwendung von voller PL keine Ausdrucksmächtigkeit gegenüber quantorenfreier Prädikatenlogik.

Wir werden jedoch nachweisen, daß volle PL echt ausdrucksmächtiger ist, wenn keine Hilfsfunktionen zugelassen werden. Allerdings können auch mit voller Prädikatenlogik nicht alle berechenbaren Datentypen monomorph spezifiziert werden, wie in [Ber93, Sch97a, Sch97b] bewiesen wurde.

Die vorliegende Arbeit ist folgendermaßen unterteilt:

Kapitel 2 enthält alle benötigten Grundlagen über Logik und Spezifikation. Dies sind einerseits Definitionen der verwendeten Begriffe und Symbole, andererseits auch einfache, allgemein bekannte Fakten.

In Kapitel 3 untersuchen wir als ersten aussichtsreichen Kandidaten für eine nicht quantorenfrei spezifizierbare Funktion das Primzahlprädikat auf den natürlichen Zahlen und weisen mit Hilfe von Resultaten aus der Berechenbarkeitstheorie dessen quantorenfreie Spezifizierbarkeit nach. Die dort bewiesene allgemeinere Aussage wird in Kapitel 6 Verwendung finden und zeigt außerdem, daß die Konstruktion eines geeigneten Beispiels nicht trivial ist.

Kapitel 4 enthält das zentrale Resultat dieser Arbeit, nämlich den Satz über die Existenz von Datentypen, die zwar mit, nicht jedoch ohne Quantoren spezifizierbar sind. Als Beispiel wählen wir endliche Graphen mit einem Vollständigkeitsprädikat und weisen die behaupteten Eigenschaften nach.

Nachdem wir gezeigt haben, daß volle Prädikatenlogik ausdrucksstärker ist als quantorenfreie, wollen wir in Kapitel 5 eine feinere Struktur innerhalb der Prädikatenlogik untersuchen. Wir zeigen, daß man nicht nur durch das allgemeine Zulassen von Quantoren an Ausdrucksstärke gewinnt, sondern durch jeden einzelnen Quantor. Die Mengen der mit k Quantoren spezifizierbaren Datentypen bilden also eine echte Hierarchie in Abhängigkeit von k.

In Kapitel 6 wollen wir uns der Frage nähern, inwieweit sich notwendige oder hinreichende Bedingungen für quantorenfreie Spezifizierbarkeit angeben lassen. Wir werden für einige (kleinere) Klassen von Datentypen konkrete Resultate nachweisen. Andererseits werden wir andeuten, daß Ansätze zur Charakterisierung, die bei ähnlichen Problemstellungen erfolgreich waren, in unserem Fall nicht in der gleichen Form greifen. Dabei widmen wir uns insbesondere auch dem Bezug zur Spezifikation unter initialer Semantik (d.h. es gelten genau die aus einer Spezifikation folgerbaren Gleichungen).

Schließlich fassen wir in Kapitel 7 die Ergebnisse dieser Arbeit zusammen, ziehen Vergleiche zu verwandten Arbeiten und zeigen noch offene Fragestellungen auf.

Kapitel 2

Grundlagen

In diesem Kapitel wollen wir die Begriffe und elementaren Sachverhalte präsentieren, auf denen die vorliegende Arbeit aufbaut. Dieses Kapitel entstand in enger Anlehnung an [Sch98], der Inhalt entstammt (bis auf geringfügige Anpassungen) den Lehrbüchern [EM85, LEW96] bzw. Artikeln [MG85, BT87, Wir90]. Diese gelten im Bereich der algebraischen Spezifikation als Standard.

Während Abschnitt 2.1 mehrsortige Algebren, ein Mittel zur mathematischen Modellierung abstrakter Datentypen, einführt, werden in Abschnitt 2.2 die der formalen Beschreibung solcher Datentypen dienenden algebraischen Spezifikationen (mittels Prädikatenlogik) erklärt.

2.1 Algebren und abstrakte Datentypen

2.1.1 Mathematische Notationen

Definition 2.1 (Mengen)
Für Mengen verwenden wir die üblichen Notationen:

- $\emptyset$ für die leere Menge
- $M \subseteq N$ für die Teilmengenbeziehung
- $M \cup N$ für die Vereinigung
- $M \cap N$ für den Durchschnitt
- $M \setminus N := \{a \in M \mid a \notin N\}$ für die Mengendifferenz
- $\overline{M}$ für das Komplement von M, falls aus dem Zusammenhang klar ist, bezüglich welcher Menge das Komplement gebildet wird.

- $M_1 \times \cdots \times M_k := \{(a_1, \ldots, a_k) \mid a_i \in M_i, 1 \leq i \leq k\}$ für das kartesische Produkt der Mengen $M_1, \ldots, M_k$.

$\square$

Definition 2.2 (Zahlen)
Ebenfalls die üblichen Bezeichnungen verwenden wir für folgende Mengen:

- natürliche Zahlen:
  $\mathbb{N} := \{0, 1, 2, \ldots\}$
- Boolesche Wahrheitswerte:
  $\mathbb{B} := \{\mathit{true}, \mathit{false}\}$.

$\square$

Definition 2.3 (Familien)
Zu einer (Index-)Menge I besteht eine **I-indizierte Familie** $M = \{M_i\}_{i \in I}$ aus je einem Objekt M_i für jedes $i \in I$. $\square$

Definition 2.4 (Wörter)
Zu einer Menge M ist M^* die Menge aller endlichen **Wörter** über M mit dem leeren Wort ϵ. $\square$

Definition 2.5 (Funktionen)
Für eine (partielle) Funktion $f : M \to N$ ist erklärt:

- $f(x) \downarrow$ steht dafür, daß der Funktionswert von f an der Stelle x **definiert** ist, $f(x) \uparrow$ dafür, daß f an der Stelle x **undefiniert** ist.

  Als Abkürzung für „$f(x) \downarrow$ und $f(x) = y$" verwenden wir $f(x) \downarrow= y$.
- Definitionsbereich (*domain*) $\mathit{dom}(f) := \{a \in M \mid f(a) \downarrow\}$
- Ist $\mathit{dom}(f) = M$, so heißt die Funktion f **total**.
- Wertebereich (*range*) $rg(f) := \{b \in N \mid \text{es gibt } a \in M \text{ mit } f(a) \downarrow= b\}$
- Graph $\mathit{graph}(f) := \{(a, b) \in M \times N \mid f(a) \downarrow= b\}$.

$\square$

Definition 2.6 (Äquivalenzklassen, Quotienten)
Ist $\sim \subseteq M \times M$ eine Äquivalenzrelation auf einer Menge M (d.h. reflexiv, symmetrisch und transitiv), so ist der **Quotient** $M/_\sim$ von M bzgl. $\sim$ definiert durch

$$M/_\sim \quad := \quad \left\{[x]_\sim \,\middle|\, x \in M\right\}$$

wobei $[x]_\sim := \{y \mid y \sim x\}$ die **Äquivalenzklasse** von x ist. $\square$

2.1.2 Signaturen und Algebren

Mehrsortige Algebren sind im wesentlichen Mengen mit darauf erklärten Funktionen. Signaturen dienen dazu, Namen für diese Mengen und Funktionen zu fixieren.

Definition 2.7 (Signaturen)
Eine **Signatur** $\Sigma = (S, F)$ besteht aus einer endlichen[1] Menge S (von **Sorten**) und einer $S^* \times S$-indizierte Familie $F = \{F_{w,s}\}_{(w,s) \in S^* \times S}$ disjunkter endlicher Mengen (von **Funktionssymbolen**).

Statt $f \in F_{w,s}$ wird manchmal die Schreibweise $(f : w \to s) \in F$ verwendet. □

Definition 2.8 (Teilsignaturen)
Gilt für zwei Signaturen $\Sigma = (S, F)$ und $\Sigma' = (S', F')$, daß $S \subseteq S'$ und $F \subseteq F'$ (genauer: $\{F_{w,s}\}_{(w,s) \in S^* \times S} \subseteq \{F'_{w,s}\}_{(w,s) \in S^* \times S}$), so nennt man Σ eine **Teilsignatur** von Σ' und schreibt $\Sigma \subseteq \Sigma'$. □

Algebren weisen Signaturen Bedeutung zu: genauer werden den Sortensymbolen (Träger-)Mengen und den Funktionssymbolen Funktionen zugeordnet.

Definition 2.9 (Σ-Algebren)
Sei $\Sigma = (S, F)$ eine Signatur. Eine **Σ-Algebra** $A = \left(\{s^A\}_{s \in S}, \{f^A\}_{f \in F} \right)$ besteht aus einer S-indizierten Familie nichtleerer **(Träger-)**Mengen s^A und aus einer F-indizierten Familie totaler Funktionen $f^A : s_1^A \times \cdots \times s_k^A \to s^A$ für $(f : s_1 \cdots s_k \to s) \in F$.

Die Klasse aller Σ-Algebren bezeichnen wir mit $Alg(\Sigma)$. □

Definition 2.10 (endliche Algebren)
Eine Σ-Algebra zur Signatur $\Sigma = (S, F)$ heißt **endlich**, wenn sie nur endlich viele Trägerelemente besitzt, d.h. falls $|s^A| < \infty$ für alle $s \in S$. □

2.1.3 Homomorphismen und Kongruenzen

Σ-Homomorphismen bilden Datenobjekte einer Σ-Algebra derart auf eine andere Σ-Algebra ab, daß die Deutung der Funktionssymbole erhalten bleibt.

[1] Im Hinblick auf den Praxisbezug sind in dieser Arbeit nur *endliche* Signaturen und *endliche* Spezifikationen relevant.

Definition 2.11 (Σ-Homomorphismen, Σ-Isomorphismen)
Seien $\Sigma = (S, F)$ und $A, B \in Alg(\Sigma)$. Ein **Σ-Homomorphismus** $h : A \to B$ ist eine S-indizierte Familie $h = \{h_s\}_{s \in S}$ totaler Abbildungen $h_s : s^A \to s^B$, so daß für jedes $(f : s_1 \cdots s_k \to s) \in F$ und alle $a_1 \in s_1^A, \ldots, a_k \in s_k^A$ gilt:

$$h_s(f^A(a_1, \ldots, a_k)) \quad = \quad f^B(h_{s_1}(a_1), \ldots, h_{s_k}(a_k)).$$

Ein **Σ-Isomorphismus** ist ein bijektiver Σ-Homomorphismus.

Zwei Algebren $A, B \in Alg(\Sigma)$ heißen **isomorph**, in Zeichen $A \cong B$, falls es einen Σ-Isomorphismus $h : A \to B$ gibt. □

Eine Kongruenzrelation auf einer Algebra ist eine Familie von Äquivalenzrelationen, welche mit den Funktionen verträglich ist (*substitutivity condition*):

Definition 2.12 (Σ-Kongruenzen)
Sei $\Sigma = (S, F)$ und $A \in Alg(\Sigma)$. Eine Familie $\sim = \{\sim_s\}_{s \in S}$ von Äquivalenzrelationen $\sim_s \subseteq s^A \times s^A$ heißt **Σ-Kongruenz auf** A, falls für alle $(f : s_1 \cdots s_k \to s) \in F$ und alle $a_1, b_1 \in s_1^A, \ldots, a_k, b_k \in s_k^A$ gilt:

$$a_1 \sim_{s_1} b_1, \ldots, a_k \sim_{s_k} b_k \quad \Rightarrow \quad f^A(a_1, \ldots, a_k) \sim_s f^A(b_1, \ldots, b_k).$$

□

2.1.4 Terme und deren Auswertung

Zunächst erklären wir die Syntax von Termen.

Definition 2.13 (Σ(*X*)-Terme)
Zu einer Signatur $\Sigma = (S, F)$ sei eine Familie $X = \{X_s\}_{s \in S}$ disjunkter Mengen (von **Variablen**) gegeben, die ebenfalls disjunkt zu den Symbolen in F sind.

Dann bezeichnet $T(\Sigma(X))_s$ die **Σ(*X*)-Terme** der Sorte $s \in S$. Dabei ist $T(\Sigma(X))_s$ die kleinste Menge mit:

- $X_s \cup F_{\lambda,s} \subseteq T(\Sigma(X))_s$,
- Mit $(f : s_1 \cdots s_k \to s) \in F$ und $t_1 \in T(\Sigma(X))_{s_1}, \ldots, t_k \in T(\Sigma(X))_{s_k}$ ist $f(t_1, \ldots, t_k) \in T(\Sigma(X))_s$.

$T(\Sigma(X)) := \{T(\Sigma(X))_s\}_{s \in S}$ ist die Familie aller Terme zur Signatur Σ. Für variablenfreie Terme, auch **Grundterme** genannt, schreiben wir im folgenden kurz $T(\Sigma) := T(\Sigma(\emptyset))$, wobei $\emptyset$ die leere Variablenfamilie ist.

Ist die Signatur Σ aus dem Kontext klar, so bezeichnet $T(f_1, \ldots, f_n)$ die Menge der $\Sigma(X)$-Terme, in denen keine anderen als die Funktionssymbole $f_1, \ldots, f_n \in \bigcup_{(w,s) \in S^* \times S} F_{w,s}$ vorkommen. $\Box$

Wir wollen noch einige Konventionen vereinbaren, um im weiteren auf langatmige Formulierungen verzichten zu können:

- t bezeichnet stets einen Term. Soll die Sorte des Terms festgelegt werden, so geschieht dies, indem der Anfangsbuchstabe des Sortenbezeichners als oberer Index verwendet wird. Ist eine Verwechslung mit einem Bezeichner für Sorten ausgeschlossen, so verwenden wir auch s für Terme, inbesondere, um die Untertermeigenschaft zu betonen.
- Auch bei Variablen x verwenden wir obere Indizes, um die Sorte zu bezeichnen.
- Ist s Unterterm von t, so bezeichnen wir dies mit $s \preceq t$.
- Ist t_1 Unterterm von t_2 oder t_2 Unterterm von t_1, so schreiben wir dafür $t_1 \asymp t_2$.
- Mit dem Symbol „$\equiv$" bezeichnen wir die syntaktische Gleichheit von Termen oder Formeln.

Insbesondere für Induktionsbeweise wird das Maß der Termtiefe oft nützlich sein:

Definition 2.14 (Termtiefe)
Die **(Schachtelungs-)Tiefe** $depth(t)$ eines Terms $t \in T(\Sigma(X))$ wird induktiv definiert durch

- $depth(x) := 0$ für $x \in X_s$,
- $depth(f) := 0$ für $f \in F_{\lambda,s}$,
- $depth(f(t_1, \ldots, t_k)) := \max\{depth(t_1), \ldots, depth(t_k)\} + 1$
für $(f : s_1 \cdots s_k \to s) \in F$ und $t_1 \in T(\Sigma(X))_{s_1}, \ldots, t_k \in T(\Sigma(X))_{s_k}$.

$\Box$

Definition 2.15 (genug Terme)
Gibt es für eine Signatur $\Sigma = (S, F)$ mit zugehöriger Variablenfamilie X zu jeder Sorte mindestens einen Term, d.h. $T(\Sigma(X))_s \neq \emptyset$ für alle $s \in S$, so sagt man, daß $\Sigma(X)$ **genug Terme**[2] hat.

[2] Die Bezeichnung „*genug Terme*" ist aus [Rei91, DEFINITION 2.2.5] übernommen.

Die Signatur Σ **hat genug Terme** (*is sensible* [Wir90, p. 681]), falls $\Sigma(\emptyset)$ genug (Grund-)Terme hat. □

Faktum 2.16 ($T(\Sigma(X))$ ist eine Σ-Algebra)
Sei eine Signatur $\Sigma = (S, F)$ mit zugehöriger Variablenfamilie X gegeben. Falls $\Sigma(X)$ genug Terme hat, so bilden die Terme $T(\Sigma(X))$ eine Σ-Algebra, die sogenannte **Termalgebra**, *durch:*

- $s^{T(\Sigma(X))} := T(\Sigma(X))_s$ *für* $s \in S$ *und*
- $f^{T(\Sigma(X))}(t_1, \ldots, t_k) := f(t_1, \ldots, t_k)$ *für* $(f : s_1 \cdots s_k \to s) \in F$ *und* $t_1 \in T(\Sigma(X))_{s_1}, \ldots, t_k \in T(\Sigma(X))_{s_k}$.

Um Termen eine Semantik zuzuordnen, wird dies zunächst für Variablen getan.

Definition 2.17 (Variablenbelegungen)
Sei $\Sigma = (S, F)$ eine Signatur, $X = \{X_s\}_{s \in S}$ eine zugehörige Familie von Variablen und $A \in Alg(\Sigma)$. Eine **Variablenbelegung** $v : X \to A$ ist eine Familie $v = \{v_s\}_{s \in S}$ totaler Abbildungen $v_s : X_s \to s^A$.

Die simultane Abänderung einer Variablenbelegung v für die (paarweise verschiedenen) Variablen $x_1 \in X_{s_1}, \ldots, x_k \in X_{s_k}$ zu den neuen Werten $a_1 \in s_1^A, \ldots, a_k \in s_k^A$ wird mit $v[x_1 \leftarrow a_1, \ldots, x_k \leftarrow a_k]$ bezeichnet. □

Definition 2.18 (Termauswertung)
Sei eine Signatur Σ und eine zugehörige Variablenfamilie derart gegeben, daß $\Sigma(X)$ genug Terme hat. Jede Variablenbelegung v läßt sich dann in eindeutiger Weise zu einem Σ-Homomorphismus $v^\# : T(\Sigma(X)) \to A$ fortsetzen, der sogenannten **Auswertung in** A **unter** v. Für einen Term $t \in T(\Sigma(X))_s$ (einer Sorte $s \in S$) schreiben wir $(t)^{v,A}$ statt $v_s^\#(t)$. Ist t ein Grundterm, so hängt seine Auswertung nicht von der Variablenbelegung v ab und man schreibt dann einfach t^A. □

2.1.5 Signaturmorphismen und Redukte

Signaturmorphismen bilden die Sorten und Funktionssymbole einer Signatur derartig auf Sorten und Funktionssymbole einer anderen Signatur ab, daß die Typen erhalten bleiben.

Definition 2.19 (Signaturmorphismen)
Für zwei Signaturen $\Sigma = (S, F)$ und $\Sigma' = (S', F')$ besteht ein **Signaturmorphismus** $\sigma : \Sigma \to \Sigma'$ aus einem Paar $\sigma = (\sigma_S, \sigma_F)$ totaler Abbildungen $\sigma_S : S \to S'$ und $\sigma_F : \bigcup_{(w,s)\in S^* \times S} F_{w,s} \to \bigcup_{(w',s')\in S'^* \times S'} F'_{w',s'}$, so daß für alle $(f : s_1 \cdots s_k \to s) \in F$ gilt:

$$\Big((\sigma_F(f) : \sigma_S(s_1) \cdots \sigma_S(s_k) \to \sigma_S(s)\Big) \quad \in \quad F'.$$

Ist $\Sigma \subseteq \Sigma'$, so bezeichnet man den Signaturmorphismus $\sigma : \Sigma \to \Sigma'$ mit

$$\begin{array}{rcll} \sigma_S(s) & = & s & \text{für alle } s \in S \\ \sigma_F(f) & = & f & \text{für alle } f \in \bigcup_{(w,s)\in S^*\times S} F_{w,s} \end{array}$$

auch als **Einbettung**. □

Mittels Reduktbildung (als Inverse von Signaturmorphismen) lassen sich Algebren und Homomorphismen einer Signatur auf eine andere Signatur übertragen.

Definition 2.20 (σ-Redukte, Σ-Redukte)
Zu einer Signatur $\Sigma = (S, F)$, einem Signaturmorphismus $\sigma : \Sigma \to \Sigma'$ und einer Σ'-Algebra A ist das σ-**Redukt** von A, in Zeichen $A|_\sigma$, die durch

- $s^{A|_\sigma} := \sigma_S(s)^A$ für $s \in S$ und
- $f^{A|_\sigma} := \sigma_F(f)^A$ für $f \in F$

erklärte Σ-Algebra.

Ist $\Sigma \subseteq \Sigma'$ so ist das Σ-**Redukt** $A|_\Sigma$ von A das durch die zugehörige Einbettung $\sigma : \Sigma \to \Sigma'$ definierte σ-Redukt, d.h. $A|_\Sigma := A|_\sigma$.

In ähnlicher Weise ist zu einem Σ'-Homomorphismus $h : A \to B$ das σ-**Redukt** von h, in Zeichen $h|_\sigma$, als der durch $(h|_\sigma)_s := h_{\sigma_S(s)}$ für $s \in S$ erklärte Σ-Homomorphismus $h|_\sigma : A|_\sigma \to B|_\sigma$ definiert. □

Definition 2.21 (Signaturmorphismen auf Termen)
Seien Signaturen $\Sigma = (S, F)$, $\Sigma' = (S', F')$ und zugehörige Variablenfamilien $X = \{X_s\}_{s\in S}$, $X' = \{X'_{s'}\}_{s'\in S'}$ gegeben. Ein Signaturmorphismus $\sigma : \Sigma \to \Sigma'$ wird dann durch eine Funktion $\sigma_X : \bigcup_{s\in S} X_s \to \bigcup_{s'\in S'} X'_{s'}$ auf Variablen erweitert, wobei für alle $s \in S$ stets $\sigma_X(X_s) \subseteq X'_{\sigma_S(s)}$ gelte.

Damit ist eine Abbildung $\sigma_T : \bigcup_{s\in S} T(\Sigma(X))_s \to \bigcup_{s'\in S'} T(\Sigma'(X'))_{s'}$ wie folgt erklärt:

- $\sigma_T(x) := \sigma_X(x)$ für Variablen $x \in \bigcup_{s\in S} X_s$
- $\sigma_T(f) := \sigma_F(f)$ für Variablen $f \in \bigcup_{s\in S} F_{\lambda,s}$
- $\sigma_T(f(t_1,\ldots,t_k)) := \sigma_F(f)(\sigma_T(t_1),\ldots,\sigma_T(t_k))$
  für $(f : s_1 \cdots s_k \to s) \in F$.

Statt σ_S, σ_F, σ_X oder σ_T schreiben wir manchmal einfach σ. □

2.1.6 Termerzeugte Algebren

Zugriff auf Objekte einer Algebra geschieht nur über Grundterme. Somit sind solche Trägerelemente, die nicht durch Grundterme angesprochen werden können, für den Benutzer des Datentyps in gewissem Sinne nutzlos (weshalb man sie auch als *junk* bezeichnet). Termerzeugten Algebren kommt eine besondere Bedeutung zu, weil sie keinen solchen *junk* enthalten. Zudem gelten für diese Klasse von Algebren verschiedene schöne Eigenschaften.

Definition 2.22 (Termerzeugtheit)
Sei $\Sigma = (S, F)$ eine Signatur und $F_c \subseteq F$ die Menge der **Konstruktoren**.

Eine Σ-Algebra $A \in Alg(\Sigma)$ heißt **termerzeugt durch** F_c, falls zu jeder Sorte $s \in S$ und jedem Trägerelement $a \in s^A$ ein Grundterm $t \in T(\Sigma)_s$ mit Funktionssymbolen ausschließlich aus F_c existiert, so daß $t^A = a$. Die Klasse aller durch F_c termerzeugten Σ-Algebren wird mit $Gen(\Sigma, F_c)$ bezeichnet.

Eine Σ-Algebra A heißt **termerzeugt** (*term-generated* [BBB+85], *reachable* [SW83], *has no junk* [MG85]), falls sie termerzeugt durch F ist. □

Bemerkung 2.23 (Termerzeugtheit)
Die Klasse der termerzeugten Σ-Algebren (Σ-*computation structures* [Wir90]) bezeichnet man kurz mit $Gen(\Sigma) := Gen(\Sigma, F)$. Man beachte, daß es — da wir nicht-leere Trägermengen fordern — termerzeugte Algebren nur für solche Signaturen geben kann, die genug Terme haben. □

Die frei erzeugten Σ-Algebren sind ein Spezialfall der termerzeugten.

Definition 2.24 (freie Termerzeugtheit)
Eine Σ-Algebra $A \in Alg(\Sigma)$ heißt **frei erzeugt** (*freely generated*), falls sie termerzeugt ist und für je zwei Grundterme $t_1, t_2 \in T(\Sigma)$ der gleichen Sorte gilt:

$$t_1^A = t_2^A \quad \Leftrightarrow \quad t_1 \equiv t_2.$$

Offensichtlich ist dies genau dann der Fall, wenn A isomorph zur Grundtermalgebra ist, d.h. $A \cong T(\Sigma)$. □

2.1.7 Berechenbare Algebren

Der Begriff der Berechenbarkeit von Algebren fußt auf dem der berechenbaren Funktion.

Definition 2.25 (Berechenbarkeit, Entscheidbarkeit)
Eine totale (oder partielle) Funktion f über den natürlichen Zahlen heißt **(effektiv) berechenbar**, falls es einen Algorithmus gibt, der die Werte von f berechnet. Die sogenannte *Churchsche These* identifiziert diesen intuitiven Begriff mit präzisen mathematischen Begriffen wie Turing-Berechenbarkeit oder Lambda-Definierbarkeit (vgl. z.B. [Kle81]).

Die Menge aller berechenbaren Funktionen von $\mathbb{N}^k$ nach $\mathbb{N}$ bezeichnen wir kurz mit P_k, die Menge aller totalen berechenbaren Funktionen von $\mathbb{N}^k$ nach $\mathbb{N}$ mit R_k.

Eine Menge $M \subseteq \mathbb{N}$ heißt **(effektiv) entscheidbar**, falls ihre *charakteristische Funktion* $\chi_M : \mathbb{N} \to \mathbb{N}$, definiert durch

$$\chi_M(n) \quad := \quad \begin{cases} 1 & \text{, falls } n \in M \\ 0 & \text{, sonst} \end{cases}$$

berechenbar ist. Für entscheidbare Mengen $M, N \subseteq \mathbb{N}$ heißt eine Funktion $f : M \to N$ **berechenbar**, falls f die Einschränkung einer berechenbaren Funktion $g : \mathbb{N} \to \mathbb{N}$ ist.

Dieser Begriff läßt sich auf Mengen $M \subseteq \mathbb{N}^k$ übertragen, indem man mittels der bekannten Aufzählungsverfahren eine 1-1-Beziehung zwischen $\mathbb{N}^k$ und $\mathbb{N}$ herstellt. □

Die folgenden Definitionen dienen nun dazu, einen Berechenbarkeitsbegriff für Algebren zu erklären. Sie stammen aus dem Bereich der konstruktiven Algebra und gehen auf Arbeiten von Rabin [Rab60] und Malćev [Mal61] zurück.

Definition 2.26 (Zahlen-Algebren)
Eine Σ-Algebra $A = \Big(\{s^A\}_{s \in S}, \{f^A\}_{f \in F} \Big)$ heißt **Zahlen-Algebra**, falls ihre Trägermengen s^A ausschließlich Teilmengen der natürlichen Zahlen $\mathbb{N}$ sind. Eine Zahlen-Algebra heißt **berechenbar**, falls alle ihre Trägermengen s^A entscheidbar und alle Funktionen f^A berechenbar sind. □

Um den Begriff der Berechenbarkeit auf beliebige Algebren (d.h. nicht nur Zahlen-Algebren) zu übertragen, bedient man sich sogenannter Koordinatisierungen (*coordinatizations*).

Definition 2.27 (Koordinatisierungen)
Sei $\Sigma = (S, F)$. Eine **Koordinatisierung** (h, B) zu einer Σ-Algebra A ist erklärt durch einen surjektiven Σ-Homomorphismus $h : B \to A$ von einer Zahlen-Algebra $B \in Alg(\Sigma)$ nach A.

Eine Koordinatisierung (h, B) heißt **berechenbar**, falls B eine berechenbare Zahlen-Algebra ist und für je zwei natürliche Zahlen $n_1, n_2 \in \mathbb{N}$ entscheidbar ist, ob $h(n_1) = h(n_2)$. $\square$

Definition 2.28 (berechenbare Algebren)
Eine Σ-Algebra A heißt **berechenbar**, falls es zu A eine berechenbare Koordinatisierung gibt.[3] $\square$

Die folgende wichtige (Invarianz-) Eigenschaft erlaubt es, den Berechenbarkeitsbegriff auch auf Isomorphieklassen $[A]_{\cong}$ von Algebren A zu übertragen.

Faktum 2.29 (Invarianz der Berechenbarkeit)
Ist eine Σ-Algebra berechenbar, so auch jede zu ihr isomorphe Σ-Algebra.

Neben anderen Charakterisierungen (vgl. [MG85, THEOREM 37]) berechenbarer Algebren ist die folgende sehr nützlich ([BT87, LEMMA 3.5]):

Faktum 2.30 (Repräsentation berechenbarer Algebren)
Jede berechenbare Σ-Algebra ist isomorph zu einer berechenbaren Zahlen-Algebra.

Für termerzeugte Algebren läßt sich die Berechenbarkeit auch durch die Entscheidbarkeit von Gleichungen ausdrücken (s. [Wir90, THEOREM 2.4.4] oder [BT83, p. 367]):

Faktum 2.31 (termerzeugte berechenbare Algebren)
Eine termerzeugte Σ-Algebra $A \in Gen(\Sigma)$ ist genau dann berechenbar, falls für je zwei Terme t_1, t_2 entscheidbar ist, ob $t_1^A = t_2^A$. [4]

---

[3] Man beachte, daß für Zahlen-Algebren die Definitionen der Berechenbarkeit nach 2.26 und 2.28 übereinstimmen.

[4] Durch Gödelnumerierungen läßt sich der Begriff der Entscheidbarkeit (von Relationen auf natürlichen Zahlen) auf Relationen auf Grundtermen übertragen.

2.1.8 Abstrakte Datentypen

Insbesondere im Bereich der Software-Spezifikation ist es oftmals nützlich, von der konkreten Repräsentation der Daten zu abstrahieren: statt einzelner Algebren (welche jeweils einer speziellen Darstellung entsprechen) betrachtet man deshalb die Klasse aller möglichen Darstellungen, d.h. die Klasse aller zu einer möglichen Darstellung isomorphen Algebren (vgl. [GTW78]).

Darüberhinaus ist es gerade bei der Anforderungsspezifikation bisweilen angemessen, von unwichtigen Eigenschaften zu abstrahieren. Beispielsweise genügt es, beim Entwurf von Sortierverfahren zu verlangen, daß auf den zu sortierenden Objekten eine Ordnung erklärt ist; eine nähere Kenntnis der Objekte selbst ist nicht vonnöten, d.h. man kann das Sortieren abstrakt für beliebige geordnete Mengen, die man dann als *einen* (*polymorphen*) abstrakten Datentyp auffaßt, definieren. Oder aber man will sich — etwa um Freiheiten zu lassen, die effiziente Implementierungen erlauben — nicht festlegen, welche Zahl oder welcher Fehlerwert geliefert werden soll, wenn eine natürliche Zahl von einer kleineren subtrahiert wird. Hier sind dann verschiedene, nicht zueinander isomorphe Realisierungen zulässig.

Diese Überlegungen motivieren den Begriff des abstrakten Datentyps als eine gegen Isomorphie abgeschlossene Klasse von Algebren (vgl. [EGL89], [LEW96]).

Definition 2.32 (Abstrakte Datentypen)
Ein **(abstrakter) Datentyp** $\mathcal{C}$ zu einer Signatur Σ ist eine gegen Isomorphie abgeschlossene Klasse von Σ-Algebren, d.h. $\mathcal{C} \subseteq Alg(\Sigma)$, so daß für je zwei $A, B \in Alg(\Sigma)$ gilt:

$$\left.\begin{array}{l} A \in \mathcal{C} \\ B \cong A \end{array}\right\} \quad \Rightarrow \quad B \in \mathcal{C}.$$

Ein abstrakter Datentyp $\mathcal{C}$ heißt **monomorph**[5], falls alle seine Algebren zueinander isomorph sind.[6] Andernfalls nennt man $\mathcal{C}$ **polymorph**.

Insbesondere bezeichnet $[A] := [A]_{\cong}$ (vgl. Definition 2.6) den zu einer Algebra A gehörigen (monomorphen) abstrakten Datentyp, und wir sprechen oftmals einfach vom (abstrakten) Datentyp A. □

Alle gegen Isomorphie invarianten Merkmale von Algebren lassen sich unmittelbar auf monomorphe Datentypen übertragen:

[5][EGL89, p. 6]: „Ein großer Teil der Theorie abstrakter Datentypen konzentriert sich auf die Spezifikation monomorpher abstrakter Datentypen. [...] 'Abstrakt' bedeutet hier soviel wie 'darstellungsunabhängig'."

[6]Man beachte, daß der leere Datentyp $\mathcal{C} = \emptyset$ zugelassen und monomorph ist.

Definition 2.33 (Eigenschaften monomorpher Datentypen)
Sei $[A]$ ein monomorpher Datentyp. $[A]$ heißt **termerzeugt (in S_c durch F_c)** (bzw. **frei erzeugt**, bzw. **endlich**, bzw. **berechenbar**) falls die Algebra A die entsprechende Eigenschaft hat. □

2.2 Algebraische Spezifikationen

Während im vorangehenden Abschnitt Algebren bzw. abstrakte Datentypen definiert wurden, wird nun ein Mittel vorgestellt, diese Objekte zu charakterisieren, d.h. zu spezifizieren. Im wesentlichen bedient man sich dabei formaler Logiken.

2.2.1 Logiken

Im Vergleich zur klassischen Logik sind die zur Beschreibung von Datentypen geeigneten Logiken auf mehrsortige Strukturen verallgemeinert und verwenden meist keine Prädikatsymbole, sondern Funktionssymbole und Gleichheit. Die in dieser Arbeit betrachteten Vertreter sind

- quantorenfreie Logik (*quantifier-free logic*) QF und
- Prädikatenlogik (*predicate logic*, *first-order logic*) PL.

Diese Logiken werden durch Angabe ihrer Syntax (bestehend aus Formeln) und ihrer Semantik (in Form einer Erfüllbarkeitsrelation) definiert.

Definition 2.34 (Syntax der Prädikatenlogik, $\boldsymbol{PL(\Sigma)}$, $\boldsymbol{QF(\Sigma)}$)
Die Menge $PL(\Sigma)$ der **prädikatenlogischen Formeln über** $\Sigma = (S, F)$ (mit zugehöriger Variablenfamilie X) ist definiert als kleinste Menge mit:

- **true** $\in PL(\Sigma)$,
- für $s \in S$ und $t_1, t_2 \in T(\Sigma(X))_s$ ist $(t_1 = t_2) \in PL(\Sigma)$,
- mit $\varphi \in PL(\Sigma)$ ist auch $(\neg\, \varphi) \in PL(\Sigma)$,
- mit $\varphi_1, \varphi_2 \in PL(\Sigma)$ ist auch $(\varphi_1 \wedge \varphi_2) \in PL(\Sigma)$ und
- für $x \in \bigcup_{s \in S} X_s$ ist mit $\varphi \in PL(\Sigma)$ auch $(\exists\, x.\, \varphi) \in PL(\Sigma)$.

Die Menge $QF(\Sigma)$ der **quantorenfreien (prädikatenlogischen) Formeln über** Σ umfaßt gerade diejenigen Formeln aus $PL(\Sigma)$, in denen '$\exists$' nicht vorkommt. □

Definition 2.35 (besondere Formeln)
Die Gleichungen und die Formel **true** nennt man **atomare Formeln**. **Literale** sind atomare oder negierte atomare Formeln. Eine Formel ist in **disjunktiver Normalform**, falls sie eine Disjunktion von Konjunktionen von Literalen ist und in **konjunktiver Normalform**, falls sie eine Konjunktion von Disjunktionen von Literalen ist. Variablenfreie Gleichungen (bzw. Literale) werden auch als **Grundgleichungen** (bzw. **Grundliterale**) bezeichnet. **Geschlossene** Formeln sind Formeln ohne freie Variablen, d.h. alle vorkommenden Variablen sind durch Quantoren gebunden.

Ist eine Signatur $\Sigma = (S, F)$ aus dem Kontext klar, so bezeichnen wir mit $PL(f_1, \ldots, f_n)$ die Menge derjenigen Formeln aus $PL(\Sigma)$, in denen höchstens die Funktionssymbole $f_1, \ldots, f_n \in \bigcup_{(w,s) \in S^* \times S} F_{w,s}$ vorkommen. □

Definition 2.36 (abkürzende Formelschreibweisen)
Wir vereinbaren die klassischen abkürzenden Schreibweisen:

$$\begin{array}{lll}
\mathbf{false} & \text{für} & (\neg\, \mathbf{true}) \\
(t_1 \neq t_2) & \text{für} & (\neg\, (t_1 = t_2)) \\
(\varphi_1 \vee \varphi_2) & \text{für} & (\neg\, (\neg\, \varphi_1) \wedge (\neg\, \varphi_2)) \\
(\varphi_1 \rightarrow \varphi_2) & \text{für} & ((\neg\, \varphi_1) \vee \varphi_2) \\
(\varphi_1 \leftrightarrow \varphi_2) & \text{für} & ((\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)) \\
(\forall\, x.\, \varphi) & \text{für} & (\neg\, (\exists\, x.\, (\neg\, \varphi)))
\end{array}$$

Zudem machen wir von den üblichen Klammereinsparungsregeln Gebrauch — insbesondere sollen Quantoren schwächer binden als alle anderen Operatoren. □

Definition 2.37 (Pränex-Normalform)
Eine Formel $\psi \in PL(\Sigma)$ heißt **in Pränex-Normalform**, falls sie von der Form

$$\psi \;\; \equiv \;\; (Q_1 x_1.\, (Q_2 x_2.\, \ldots (Q_k x_k.\, (\varphi)) \ldots))$$

ist. Hierbei ist $Q_j \in \{\forall, \exists\}$ für jedes $j \in \{1, \ldots, k\}$, und φ ist quantorenfrei und in konjunktiver Normalform. φ wird auch die **Matrix** von ψ genannt. □

Definition 2.38 (Semantik)
Sei $\Sigma = (S, F)$ eine Signatur, $X = \{X_s\}_{s \in S}$ eine zugehörige Familie von Variablen, $A \in Alg(\Sigma)$ und $v : X \rightarrow A$ eine Variablenbelegung. Für Formeln $\varphi \in PL(\Sigma)$ wird ihre **Gültigkeit in** A **unter** v, in Zeichen $A, v \models \varphi$, induktiv wie folgt definiert:

$$\begin{array}{lll}
A, v \models \mathbf{true} & & \\
A, v \models (t_1 = t_2) & :\Leftrightarrow & (t_1)^{v,A} = (t_2)^{v,A} \\
A, v \models (\neg\,\varphi) & :\Leftrightarrow & A, v \not\models \varphi \\
A, v \models (\varphi_1 \wedge \varphi_2) & :\Leftrightarrow & A, v \models \varphi_1 \text{ und } A, v \models \varphi_2 \\
A, v \models (\exists\, x.\, \varphi) & :\Leftrightarrow & A, v[x \leftarrow a] \models \varphi \text{ für ein } a \in s^A \\
 & & \qquad \text{(wobei } s \text{ die Sorte von } x \text{ ist).}
\end{array}$$

Man sagt, daß φ **in** A **gilt** oder daß A die Formel φ **erfüllt**, in Zeichen $A \models \varphi$, falls $A, v \models \varphi$ für alle Variablenbelegungen v.

Für $\mathcal{C} \subseteq Alg(\Sigma)$ schreibt man $\mathcal{C} \models \varphi$, um auszudrücken, daß $A \models \varphi$ für alle $A \in \mathcal{C}$. Ist $\Phi \subseteq PL(\Sigma)$ so steht $A \models \Phi$ dafür, daß $A \models \varphi$ für alle $\varphi \in \Phi$ gilt. □

Faktum 2.39 (Existenz der Pränex-Normalform)
Zu jeder Formel $\phi \in PL(\Sigma)$ existiert eine Formel $\psi \in PL(\Sigma)$ in Pränex-Normalform, so daß für alle $A \in Alg(\Sigma)$:

$$A \models \phi \leftrightarrow \psi,$$

d.h. ϕ und ψ sind in allen Σ-Algebren äquivalent.

Definition 2.40 (abkürzende Schreibweisen bei freien Variablen)
Wir benutzen manchmal die Schreibweise $t(x_1, \ldots, x_k)$, um anzudeuten, daß in dem Term t höchstens die Variablen $x_1, \ldots, x_k$ vorkommen. Entsprechend besagt die Schreibweise $\varphi(x_1, \ldots, x_k)$, daß in der Formel φ höchstens die Variablen $x_1, \ldots, x_k$ frei vorkommen.

Für einen Term $t(x_1, \ldots, x_k)$ ist $t^A(a_1, \ldots, a_k)$ eine Kurzschreibweise für das durch $(t(x_1, \ldots, x_k))^{v[x_1 \leftarrow a_k, \ldots, x_k \leftarrow a_k], A}$ beschriebene Trägerelement (wobei die Wahl der Variablenbelegung v irrelevant ist). Wie bei Termen steht für eine Formel $\varphi(x_1, \ldots, x_k)$ die Schreibweise $A \models \varphi(a_1, \ldots, a_k)$ dafür, daß $A, v[x_1 \leftarrow a_k, \ldots, x_k \leftarrow a_k] \models \varphi(x_1, \ldots, x_k)$ für alle (oder äquivalent: für ein) v gilt. □

2.2.2 Spezifikationen

Zunächst ist eine Spezifikation — im Sinne des axiomatischen Ansatzes — einfach eine endliche Menge von Formeln zu einer bestimmten Signatur. Wie in den meisten algebraischen Spezifikationssprachen üblich, erlauben wir, zusätzlich Termerzeugtheitsforderungen auszudrücken.

Definition 2.41 (Spezifikationen)
Für $L \in \{QF, PL\}$ besteht eine **L-Spezifikation** $SP = (\Sigma, \Phi)$ aus einer Signatur $\Sigma = (S, F)$ und einer endlichen Menge $\Phi \subseteq L(\Sigma)$ von Formeln, den **Axiomen**. □

Definition 2.42 (Semantik von Spezifikationen)
Für eine **L-Spezifikation** $SP = (\Sigma, \Phi)$ definieren wir (entsprechend [Wir90, p. 697]) **lose Semantik** (*loose semantics*):
$Mod(SP) := \{A \in Gen(\Sigma) \mid A \models \Phi\}$.[7]

Die Elemente von $Mod(SP)$ nennt man **Modelle** von SP. Für $\varphi \in L(\Sigma)$ schreibt man statt $Mod(SP) \models \varphi$ einfach $SP \models \varphi$. □

Bei der losen Semantik werden in der Regel beliebige — ggf. quantorenfreie — Formeln als Axiome zugelassen.

Definition 2.43 (lose Spezifikationsmethoden)
Ein abstrakter Datentyp $\mathcal{C}$ heißt

- QF-**spezifizierbar**, falls eine QF-Spezifikation SP mit $Mod(SP) = \mathcal{C}$ existiert.
- PL-**spezifizierbar**, falls eine PL-Spezifikation SP mit $Mod(SP) = \mathcal{C}$ existiert.

□

Will man zeigen, daß eine Spezifikation SP einen monomorphen Datentyp A unter loser Semantik spezifiziert, so läßt sich das in zwei Teilaufgaben zerlegen:

(1) gilt $A \in Mod(SP)$?

(2) ist $Mod(SP)$ monomorph ?

Gilt letztere Eigenschaft, so nennt man SP selbst monomorph:

Definition 2.44 (Monomorphie)
Eine L-Spezifikation SP heißt **monomorph**, falls der Datentyp $Mod(SP)$ monomorph ist, d.h. falls je zwei Algebren in $Mod(SP)$ isomorph sind. □

[7] Man beachte, daß wir uns also von vornherein auf die termerzeugten Modelle einer Spezifikation beschränken.

Faktum 2.45 (lose Spezifizierbarkeit monomorpher Datentypen)
Ein monomorpher Datentyp $A \in Alg(\Sigma)$ ist genau dann PL-spezifizierbar (bzw. QF-spezifizierbar), wenn eine monomorphe PL-Spezifikation (bzw. eine monomorphe QF-Spezifikation) SP mit $A \in Mod(SP)$ existiert.

Zum Nachweis der Monomorphie bedient man sich oft folgender Charakterisierung (siehe z.B. [Wir90, FACT 2.3.2] oder [Rei92, THEOREM 5.5]):

Faktum 2.46 (Charakterisierung der Monomorphie)
Sei Σ eine Signatur, die genug Terme hat. Eine L-Spezifikation $SP = (\Sigma, \Phi)$ ist genau dann monomorph, wenn sie die Grundgleichungen festlegt, d.h. wenn für alle Grundgleichungen $(t_1 = t_2) \in PL(\Sigma)$ gilt, daß $SP \models t_1 = t_2$ oder $SP \models t_1 \neq t_2$.

2.2.3 Beispiele

Zum Ende dieses Kapitels wollen wir einige einfache Beispiele von Signaturen, Datentypen und Spezifikationen anführen, auf die später bisweilen verwiesen wird. Für die Darstellung von Signaturen und Spezifikationen bedienen wir uns einer an Sprachen wie `ASL` [SW83] oder `ACT-ONE` [Cla89] angelehnten Notation. Eine Signatur $\Sigma = (S, F)$ wird dabei in einem Schema der Form

signature Σ	
sorts	S
functions	F
end signature	

und eine Spezifikation $SP = (\Sigma, \Phi)$ mit $\Sigma = (S, F)$ in einem Schema der Form

specification SP	
sorts	S
functions	F
axioms	Φ
end specification	

dargestellt. Des weiteren verwenden wir noch ein **import**-Konstrukt, welches gerade einfaches syntaktisches Einfügen der Teile der jeweiligen Signatur bzw. Spezifikation bedeutet. Ist beispielsweise $\Sigma' = (S', F')$, so steht

signature Σ	
import	Σ'
sorts	S
functions	F
end signature	

für die Signatur $\Sigma = (S \cup S', F \cup F')$. Entsprechend ist zu einer Spezifikation $SP' = ((S', F'), \Phi')$ mit

specification SP	
import	SP'
sorts	S
functions	F
axioms	Φ
end specification	

die Spezifikation $SP = ((S \cup S', F \cup F'), \Phi \cup \Phi')$ gemeint.

Nun zu den Beispielen. Ganz elementar ist der endliche und frei erzeugte Datentyp **BOOL** der Wahrheitswerte. Diesen legen wir zur Signatur

signature Σ_{BOOL}			
sorts	`Bool`		
functions	`true`	:	$\to$ `Bool`
	`false`	:	$\to$ `Bool`
end signature			

durch die wie folgt definierte Σ_{BOOL}-Algebra fest:

$$\begin{aligned}
\mathtt{Bool}^{\mathbf{BOOL}} &= \mathbb{B} \\
\mathtt{true}^{\mathbf{BOOL}} &= \mathit{true} \\
\mathtt{false}^{\mathbf{BOOL}} &= \mathit{false}.
\end{aligned}$$

Die Spezifikation

$$\boxed{\begin{array}{l}
\textbf{specification}\ \mathrm{BOOL} \\
\quad \begin{array}{ll}
\textbf{sorts} & \texttt{Bool} \\
\textbf{variables} & \texttt{b} : \texttt{Nat} \\
\textbf{functions} & \texttt{true} \ : \ \to \texttt{Bool} \\
 & \texttt{false} : \ \to \texttt{Bool} \\
\textbf{axioms} & (1)\ \texttt{true} \neq \texttt{false} \\
 & (2)\ \texttt{b} = \texttt{true} \vee \texttt{b} = \texttt{false}
\end{array} \\
\textbf{end specification}
\end{array}}$$

legt diesen Datentyp unter loser Semantik fest, d.h. $\mathit{Mod}(\mathrm{BOOL}) = [\mathbf{BOOL}]$. Das zweite Axiom wird an dieser Stelle wegen der Termerzeugtheitsforderung noch nicht benötigt. Wenn wir jedoch später noch Funktionen mit Zielsorte `Bool` hinzufügen, stellt dieses Axiom sicher, daß auch deren Werte alle `true` oder `false` sind.

Ebenso elementar ist der Datentyp **NAT** der natürlicher Zahlen. Genauer meinen wir damit die zur Signatur

$$\boxed{\begin{array}{l}
\textbf{signature}\ \Sigma_{\mathrm{NAT}} \\
\quad \begin{array}{ll}
\textbf{sorts} & \texttt{Nat} \\
\textbf{functions} & \texttt{zero} : \qquad \to \texttt{Nat} \\
 & \texttt{succ} : \texttt{Nat} \to \texttt{Nat}
\end{array} \\
\textbf{end signature}
\end{array}}$$

gehörige Σ_{NAT}-Algebra **NAT** mit

$$\begin{array}{lcl}
\texttt{Nat}^{\mathbf{NAT}} & = & \mathbb{N} \\
\texttt{zero}^{\mathbf{NAT}} & = & 0 \\
\texttt{succ}^{\mathbf{NAT}}(n) & = & n+1.
\end{array}$$

Wir benötigen zwei Axiome, um die freie Erzeugtheit des Datentyps sicherzustellen. Für die Spezifikation

specification NAT		
	sorts	`Nat`
	functions	`zero :` $\rightarrow$ `Nat`
		`succ : Nat` $\rightarrow$ `Nat`
	variables	`x, y : Nat`
	axioms	(1) `succ(x)` $\neq$ `zero`
		(2) `succ(x)` $=$ `succ(y)` $\rightarrow$ `x` $=$ `y`
end specification		

erhält man also $\mathit{Mod}(\mathrm{NAT}) = [\mathbf{NAT}]$.

In Kapitel 3 werden wir Beispiele betrachten, bei denen wir auch die Addition und Multiplikation erlauben wollen. Wir vereinbaren daher schon an dieser Stelle die notwendige Signatur und Algebra und geben eine geeignete Spezifikation an. Zur Signatur

signature Σ_{MULT}		
	import	Σ_{NAT}
	functions	`add : Nat` $\times$ `Nat` $\rightarrow$ `Nat`
		`mult : Nat` $\times$ `Nat` $\rightarrow$ `Nat`
end signature		

sei **NAT-MULT** die Σ_{MULT}-Algebra mit

$$\begin{array}{rcl} \mathbf{NAT\text{-}MULT}|_{\Sigma_{\mathrm{NAT}}} & = & \mathbf{NAT} \\ \mathtt{add}^{\mathbf{NAT\text{-}MULT}}(x,y) & = & x+y \\ \mathtt{mult}^{\mathbf{NAT\text{-}MULT}}(x,y) & = & xy. \end{array}$$

Der Datentyp **NAT-MULT** ist durch die folgende Spezifikation *QF*-spezifizierbar:

specification NAT-MULT

import NAT

functions
$\texttt{add} : \texttt{Nat} \times \texttt{Nat} \to \texttt{Nat}$
$\texttt{mult} : \texttt{Nat} \times \texttt{Nat} \to \texttt{Nat}$

variables $\texttt{x}, \texttt{y} : \texttt{Nat}$

axioms
(1) $\texttt{add}(\texttt{zero}, y) = y$
(2) $\texttt{add}(\texttt{succ}(x), y) = \texttt{succ}(\texttt{add}(x, y))$
(3) $\texttt{mult}(\texttt{zero}, y) = \texttt{zero}$
(4) $\texttt{mult}(\texttt{succ}(x), y) = \texttt{add}(\texttt{mult}(x, y), y)$

end specification

Daß alle Axiome aus NAT-MULT in **NAT-MULT** gelten, kann sehr leicht nachgeprüft werden. Um zu zeigen, daß NAT-MULT alle Grundgleichungen festlegt, genügt es zu beobachten, daß Axiome (1) und (2) die Funktion `add`, Axiome (3) und (4) die Funktion `mult` durch strukturelle Induktion im ersten Argument definieren — insbesondere läßt sich also jede Grundgleichung auf eine Gleichung zwischen Termen der Form $\texttt{succ}^k(\texttt{zero})$ mit $k \in \mathbb{N}$ „reduzieren", und solche Grundgleichungen sind wegen der Monomorphie von NAT festgelegt. Also können wir mit Hilfe von Faktum 2.46 schließen, daß $\mathit{Mod}(\text{NAT-MULT}) = [\textbf{NAT-MULT}]$.

Kapitel 3

Natürliche Zahlen mit Multiplikation

Bei der Suche nach einem geeigneten Beispiel, das zwar mit, nicht jedoch ohne Quantoren spezifizierbar ist, wird man vielleicht zunächst Funktionen oder Prädikate (letztere mit Zielsorte `Bool`) auf den natürlichen Zahlen in Erwägung ziehen.[1] Zum Beispiel scheint das Primzahlprädikat, zu definieren über der Signatur

signature Σ_{PRIM}
import $\Sigma_{\mathrm{MULT}}, \Sigma_{\mathrm{BOOL}}$
functions $\mathtt{prim} : \mathtt{Nat} \rightarrow \mathtt{Bool}$
end signature

zunächst ein sehr aussichtsreicher Kandidat zu sein. Denn während die Nicht-Primzahlen (außer $0, 1$) genau Zahlen der Form $(a + 2)(b + 2)$ mit $a, b \in \mathbb{N}$ sind, ist für die Primzahlen selbst keine entsprechend einfache Formel offensichtlich. Andererseits ist ja eine Spezifikation unter Zuhilfenahme von Quantoren sehr einfach möglich durch folgendes Axiom:

$$\begin{aligned}\mathtt{prim(x)} = \mathtt{true} \leftrightarrow (&\mathtt{x} \neq \mathtt{zero} \wedge \mathtt{x} \neq \mathtt{succ(zero)}\\ &\wedge \forall \mathtt{u}, \mathtt{v}.\, \mathtt{mult(succ(succ(u)), succ(succ(v)))} \neq \mathtt{x}).\end{aligned}$$

Um so mehr mag es überraschen, daß ein solches Primzahlprädikat sehr wohl quantorenfrei spezifizierbar ist. Es gibt nämlich ein Polynom mit ganzzahli-

[1] Auch läßt das in Kapitel 4 vorgestellte Beispiel vermuten, daß das Quadratzahlprädikat über der Signatur $\Sigma = \{\mathtt{zero}, \mathtt{succ}, \mathtt{add}, \mathtt{square}\}$ nicht quantorenfrei spezifizierbar ist (eine prädikatenlogische Spezifikation ist zwar etwas trickreich, aber möglich). Allerdings dürfte der Beweis der Nicht-Spezifizierbarkeit dieses Datentyps mindestens so schwierig sein wie der aus Kapitel 4.

gen Koeffizienten, dessen positive Werte genau die Primzahlen sind.[2] Allgemein lassen sich jedes entscheidbare Prädikat und jede berechenbare Funktion auf den natürlichen Zahlen quantorenfrei spezifizieren, wenn die entsprechende Signatur die Signatur Σ_{MULT} enthält. Da dieses Ergebnis auf den ersten Blick etwas überraschen mag und außerdem als positives Resultat inmitten der Nicht-QF-Spezifizierbarkeitsresultate der nächsten Kapitel steht, wollen wir ihm einen ausführlichen Beweis widmen.

3.1 Entscheidbare Prädikate

Zu jeder Menge $M \subseteq \mathbb{N}^n$ definieren wir die Signatur

signature $\Sigma_{\mathrm{NAT_M}}$
import $\Sigma_{\mathrm{MULT}}, \Sigma_{\mathrm{BOOL}}$
functions `char` : `Nat`^{n} $\to$ `Bool`
end signature

und die $\Sigma_{\mathrm{NAT_M}}$-Algebra $\mathbf{NAT}_M$ mit $\mathbf{NAT}_M|_{\Sigma_{\mathrm{MULT}}} = \mathbf{NAT\text{-}MULT}$ und

$$\mathtt{char}^{\mathbf{NAT}_M}(x_1,\ldots,x_n) \quad = \quad \begin{cases} \mathit{true} & \text{, falls } (x_1,\ldots,x_n) \in M \\ \mathit{false} & \text{, falls } (x_1,\ldots,x_n) \notin M, \end{cases}$$

d.h. `char` wird definiert als charakteristische Funktion der Menge M. Dann gilt folgender

Satz 3.1 (Spezifizierbarkeit von $\mathbf{NAT}_M$)
Ist $M \subseteq \mathbb{N}^n$ eine entscheidbare *Menge, so gibt es eine QF-Spezifikation SP mit $Mod(SP) = [\mathbf{NAT}_M]$.*

Zum Beweis benötigen wir Resultate der Berechenbarkeitstheorie, die in engem Zusammenhang mit der Unlösbarkeit des Zehnten Hilbertschen Problems stehen. Wir wollen daher zunächst die nötigen Begriffe und Resultate vorstellen.

[2] Da die Darstellung des Polynoms etwas umfangreich ist, ist es lediglich in Anhang A aufgeführt.

Definition 3.2 (diophantische Menge)
Eine Menge $M \subseteq \mathbb{N}^n$ heißt **diophantisch**, wenn Polynome $p, q : \mathbb{N}^{n+k} \to \mathbb{N}$ mit natürlichzahligen Koeffizienten existieren, so daß

$$\begin{aligned}
&(x_1, \ldots, x_n) \in M \\
&\iff \quad \text{es gibt } y_1, \ldots, y_k \in \mathbb{N} \text{ mit} \\
&\qquad\quad p(x_1, \ldots, x_n, y_1, \ldots, y_k) = q(x_1, \ldots, x_n, y_1, \ldots, y_k).
\end{aligned}$$

□

Die zentrale Rolle beim Beweis unseres Satzes spielt der folgende

Satz 3.3 (Satz von Matiyasevich)
Ist $M \subseteq \mathbb{N}^n$ rekursiv aufzählbar, so ist M diophantisch.

Beweis. Der Beweis dieses Satzes ist zu aufwendig, als daß er an dieser Stelle wiedergegeben werden könnte. Er verwendet tiefliegende Hilfsmittel aus der Zahlentheorie und stützt sich auf umfangreiche Vorarbeiten.

Der Originalbeweis findet sich in [Mat70], gute Darstellungen werden auch in [Man77] und vor allem in [Mat93] gegeben. ■

Bemerkung 3.4 (zum Satz von Matiyasevich)
Historisch war der Satz von Matiyasevich der letzte und entscheidende Schritt zum Beweis der Unlösbarkeit des Zehnten Hilbertschen Problems. Das Zehnte Hilbertsche Problem lautete, ein Entscheidungsverfahren für die Existenz von Lösungen diophantischer Gleichungen anzugeben, und unter Verwendung des obigen Satzes konnte das Problem auf das Halteproblem für Turing-Maschinen reduziert werden. Ein großer Teil der Vorarbeit zum Beweis des obigen Satzes geht auf Davis, Putnam und Robinson zurück.

Da das Zehnte Hilbertsche Problem zum Standardumfang von Lehrbüchern zur Berechenbarkeit gehört, ist der Satz zum Beispiel auch in [Cut80] oder [Odi89] zu finden.

Der Beweis ist sogar insofern konstruktiv, als er nicht nur die Existenz entsprechender Polynome garantiert, sondern auch ein Konstruktionsverfahren für solche Polynome enthält. Allerdings sind die entsprechend konstruierten Polynome in Bezug auf Grad oder Variablenanzahl keineswegs minimal. □

Da unser Beweis sich zentral auf die Darstellung von aufzählbaren Mengen durch Polynome stützt, müssen wir Polynome über der Signatur Σ_{MULT} darstellen können. Eine entsprechende Aussage enthält das nachfolgende

Lemma 3.5 (Darstellbarkeit von Polynomen)
Ist $p : \mathbb{N}^n \to \mathbb{N}$ *ein Polynom, dessen Koeffizienten alle natürlichzahlig sind, so existiert über der Signatur* Σ_{MULT} *ein Term* $t_p(\mathtt{x}_1, \ldots, \mathtt{x}_n)$*, so daß für jede Variablenbelegung* $v : X \to \mathbf{NAT\text{-}MULT}$ *gilt:*

$$(t_p)^{v,\mathbf{NAT\text{-}MULT}} \quad = \quad p(v(\mathtt{x}_1), \ldots, v(\mathtt{x}_n)).$$

Beweis. Zum Beweis konstruiert man auf naheliegende Weise den entsprechenden Term. Die natürlichzahligen Koeffizienten a werden durch Terme der Form $\mathtt{succ}^a(\mathtt{zero})$ dargestellt, die Variablen x_j durch $\mathtt{x}_j$ selbst. Produkte und Summen werden durch `mult` und `add` beschrieben. Die behauptete Gleichheit kann dann einfach nachgerechnet bzw. formal durch einen Induktionsbeweis gezeigt werden. ■

Mit Hilfe des Satzes von Matiyasevich und des eben bewiesenen Darstellungslemmas können wir nunmehr den Beweis von Satz 3.1 in Angriff nehmen.

Beweis von Satz 3.1. Sei $M \subseteq \mathbb{N}^n$ entscheidbar. Dann ist sowohl die Menge M als auch ihr Komplement $\overline{M}$ rekursiv aufzählbar, also nach dem Satz von Matiyasevich diophantisch. Nach Definition gibt es also Polynome $p_M, q_M : \mathbb{N}^{n+k} \to \mathbb{N}$ und $p_{\overline{M}}, q_{\overline{M}} : \mathbb{N}^{n+m} \to \mathbb{N}$ mit natürlichzahligen Koeffizienten, so daß

$$\begin{aligned}
&(x_1, \ldots, x_n) \in M \\
&\quad \Longleftrightarrow \quad \text{es gibt } y_1, \ldots, y_k \text{ mit} \\
&\qquad\qquad p_M(x_1, \ldots, x_n, y_1, \ldots, y_k) = q_M(x_1, \ldots, x_n, y_1, \ldots, y_k), \\
&(x_1, \ldots, x_n) \notin M \\
&\quad \Longleftrightarrow \quad \text{es gibt } z_1, \ldots, z_m \text{ mit} \\
&\qquad\qquad p_{\overline{M}}(x_1, \ldots, x_n, z_1, \ldots, z_m) = q_{\overline{M}}(x_1, \ldots, x_n, z_1, \ldots, z_m).
\end{aligned}$$

Mit Hilfe der Terme $t_{p_M}, t_{q_M}, t_{p_{\overline{M}}}$ und $t_{q_{\overline{M}}}$ entsprechend Lemma 3.5 können wir nun eine Spezifikation NAT_M angeben:

specification NAT_M
  import $\mathrm{NAT\text{-}MULT}, \mathrm{BOOL}$
  functions $\mathtt{char} : \mathtt{Nat}^n \to \mathtt{Bool}$
  variables $\mathtt{x}_1, \ldots, \mathtt{x}_n : \mathtt{Nat}$
  $\mathtt{y}_1, \ldots, \mathtt{y}_k : \mathtt{Nat}$
  $\mathtt{z}_1, \ldots, \mathtt{z}_m : \mathtt{Nat}$
  axioms $t_{p_M}(\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{y}_1, \ldots, \mathtt{y}_k) = t_{q_M}(\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{y}_1, \ldots, \mathtt{y}_k)$
  $\to \mathtt{char}(\mathtt{x}_1, \ldots, \mathtt{x}_n) = \mathtt{true}$
  $t_{p_{\overline{M}}}(\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{z}_1, \ldots, \mathtt{z}_m) = t_{q_{\overline{M}}}(\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{z}_1, \ldots, \mathtt{z}_m)$
  $\to \mathtt{char}(\mathtt{x}_1, \ldots, \mathtt{x}_n) = \mathtt{false}$
end specification

Nach den vorangegangen Ausführungen folgt unmittelbar, daß für obige Spezifikation NAT_M gilt: $Mod(\mathrm{NAT}_M) = [\mathbf{NAT}_M]$. ■

Die Existenz einer quantorenfreien Spezifikation von des Primzahlprädikats ist natürlich nur ein Spezialfall des eben bewiesenen Satzes. Wir setzen $M := \{x \in \mathbb{N} \mid x \text{ ist prim}\}$ und wählen die Spezifikation entsprechend Satz 3.1.

In diesem speziellen Fall ist es ziemlich leicht, Polynome $p_{\overline{M}}$ und $q_{\overline{M}}$ mit den gewünschten Eigenschaften zu bestimmen. Deutlich schwieriger hingegen sind die Polynome p_M und q_M. Aus dem in Anhang A angegeben Polynom lassen sich p_M und q_M bestimmen. Die Konstruktion minimaler Polynome (bezüglich Grad oder Variablenzahl) für die Menge der Primzahlen hat in der zahlentheoretischen Literatur einige Beachtung erhalten.

3.2 Berechenbare Funktionen

Eine ähnliche Aussage wie für entscheidbare Mengen gilt auch für totale berechenbare Funktionen auf den natürlichen Zahlen, da der Graph einer solchen Funktion ja entscheidbar ist. Um den Satz formulieren zu können, definieren wir zu jeder Funktion $f : \mathbb{N}^n \to \mathbb{N}$ die Signatur

signature $\Sigma_{\mathrm{NAT_f}}$
  import Σ_{MULT}
  functions $\mathtt{f} : \mathtt{Nat}^n \to \mathtt{Nat}$
end signature

und die $\Sigma_{\mathrm{NAT_f}}$-Algebra $\mathbf{NAT}_f$ durch $\mathbf{NAT}_f|_{\Sigma_{\mathrm{MULT}}} = \mathbf{NAT\text{-}MULT}$ und

$$\mathtt{f}^{\mathbf{NAT}_f} \quad = \quad f.$$

Dann läßt sich unsere Behauptung folgendermaßen formulieren:

Satz 3.6 (Spezifizierbarkeit von $\mathbf{NAT}_f$)
Ist $f : \mathbb{N}^n \to \mathbb{N}$ *eine* totale, berechenbare *Funktion, so gibt es eine QF-Spezifikation SP mit Mod*$(SP) = [\mathbf{NAT}_f]$.

Beweis. Der Beweis verläuft sehr ähnlich wie der von Satz 3.1. Ist f total und berechenbar, so ist

$$graph(f) \quad = \quad \{(x_1, \ldots, x_{n+1}) \mid x_{n+1} = f(x_1, \ldots, x_n)\} \quad \subseteq \quad \mathbb{N}^{n+1}$$

entscheidbar, also insbesondere rekursiv aufzählbar. Unter Anwendung des Satzes von Matiyasevich erhalten wir nun, daß $graph(f)$ diophantisch ist, und es existieren demnach Polynome $p_f, q_f : \mathbb{N}^{n+k+1} \to \mathbb{N}$ mit natürlichzahligen Koeffizienten, so daß

$$\begin{aligned}
&f(x_1, \ldots, x_n) = x_{n+1} \\
&\quad \Longleftrightarrow \quad \text{es gibt } y_1, \ldots, y_k \text{ mit} \\
&\qquad\qquad p_f(x_1, \ldots, x_{n+1}, y_1, \ldots, y_k) = q_f(x_1, \ldots, x_{n+1}, y_1, \ldots, y_k).
\end{aligned}$$

Unter Verwendung der Terme t_{p_f} und t_{q_f} aus Lemma 3.5 erhalten wir die folgende Spezifikation:

specification	NAT_f
import	NAT-MULT
functions	$\mathtt{f} : \mathtt{Nat}^n \to \mathtt{Nat}$
variables	$\mathtt{x}_1, \ldots, \mathtt{x}_{n+1} : \mathtt{Nat}$ $\mathtt{y}_1, \ldots, \mathtt{y}_k : \mathtt{Nat}$
axioms	$t_p(\mathtt{x}_1, \ldots, \mathtt{x}_{n+1}, \mathtt{y}_1, \ldots, \mathtt{y}_k) = t_q(\mathtt{x}_1, \ldots, \mathtt{x}_{n+1}, \mathtt{y}_1, \ldots, \mathtt{y}_k)$ $\to \mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) = \mathtt{x}_{n+1}$
end specification	

Auch in diesem Fall ist leicht einzusehen, daß für die Spezifikation NAT_f tatsächlich $Mod(\mathrm{NAT}_f) = [\mathbf{NAT}_f]$ gilt. Insbesondere folgt aus der oben angeführten Äquivalenz, daß die Spezifikation konsistent ist — zu jedem $(x_1, \ldots, x_n)$ existiert ja genau ein x_{n+1}, so daß die linke Seite der Implikation in obigem Axiom erfüllt sein kann. ∎

Kapitel 4

Vollständigkeit endlicher Graphen

In diesem Kapitel wollen wir zeigen, daß es Datentypen gibt, die zwar PL-spezifizierbar, nicht jedoch QF-spezifizierbar sind, d.h. zu deren Spezifikation Quantoren vonnöten sind.

Das Beispiel, das wir nachfolgend betrachten wollen, ist die Vollständigkeit endlicher Graphen, in dem Sinne, daß je zwei nicht-isolierte Knoten miteinander verbunden sind. Wir werden für dieses Prädikat eine PL-Spezifikation angeben und anschließend nachweisen, daß keine QF-Spezifikation existiert. Hierzu soll zunächst präzisiert werden, was wir unter Graphen verstehen wollen; ferner sollen die Signatur und einige abkürzende Schreibweisen eingeführt werden.

4.1 Definition von Graphen

Wir wollen im folgenden unter einem *Graphen* stets einen gerichteten, ungewichteten Graphen mit endlich vielen Kanten verstehen, dessen Knotenmenge die natürlichen Zahlen sind. Da in unserem Fall die Knotenmenge für jeden Graphen gleich ist, können wir einen Graphen mit der Menge seiner Kanten identifizieren. Ein Graph ist also eine endliche Teilmenge von $\mathbb{N} \times \mathbb{N}$.

Jede solche endliche Menge läßt sich durch endlich häufiges Hinzunehmen jeweils einer Kante (eines Paares) zur ursprünglich leeren Menge erzeugen, d.h. zur Darstellung aller Graphen sind lediglich eine konstante Funktion für den leeren Graphen und eine Funktion zum Hinzunehmen von Kanten vonnöten.

Obwohl wir Graphen als endliche Mengen von *Paaren* natürlicher Zahlen definiert haben, wird es sich im folgenden häufig als günstig erweisen, über Knoten des Graphen zu sprechen. Wir wollen sagen, ein Knoten $x \in \mathbb{N}$ befinde sich im Graphen G, falls x Anfangs- oder Endpunkt einer Kante von G ist.

Definition 4.1 (Graphen)
Die eben informell eingeführten Begriffe werden nochmals in folgenden Definitionen zusammengefaßt:

1. Ein **Graph** G ist eine endliche Teilmenge $G \subset \mathbb{N} \times \mathbb{N}$. Die Menge aller Graphen bezeichnen wir mit $\mathcal{G}$.

2. Ein Knoten $x \in \mathbb{N}$ heiße **in** G $(x \in G)$, falls ein $y \in \mathbb{N}$ existiert mit

$$(x, y) \in G \quad \text{oder} \quad (y, x) \in G.$$

   Andernfalls heißt x **isoliert** in G.

□

Nachdem wir beschrieben haben, was wir unter Graphen verstehen wollen, sollen nun die Signatur Σ_{GRAPH} und der Datentyp **GRAPH** angegeben werden, die nachfolgend die Grundlage der Beweise bilden werden.

signature Σ_{GRAPH}		
import	Σ_{NAT}	
sorts	`Graph`	
functions	`empty`	$: \quad \rightarrow$ `Graph`
	`connect`	$:$ `Nat` $\times$ `Nat` $\times$ `Graph` $\rightarrow$ `Graph`
end signature		

Der Datentyp **GRAPH** wird zur Signatur Σ_{GRAPH} erklärt durch

$$\mathbf{GRAPH}|_{\Sigma_{\mathrm{NAT}}} = \mathbf{NAT}$$

und die folgenden Definitionen:

$$\begin{aligned}
\mathtt{Graph}^{\mathbf{GRAPH}} &= \mathcal{G} \\
\mathtt{empty}^{\mathbf{GRAPH}} &= \emptyset \\
\mathtt{connect}^{\mathbf{GRAPH}}(x, y, G) &= G \cup \{(x, y)\}
\end{aligned}$$

Im folgenden werden wir zeigen, daß die Eigenschaft eines Graphen, vollständig zu sein, zwar PL-spezifizierbar, nicht jedoch QF-spezifizierbar ist. Dabei definieren wir die Vollständigkeit eines Graphen folgendermaßen:

Definition 4.2 (Vollständigkeit von Graphen)
Ein Graph $G \subset \mathbb{N} \times \mathbb{N}$ heiße *vollständig*, wenn für alle $x, y \in G$ gilt, daß $(x, y) \in G$. Wir bezeichnen mit $\mathcal{V}$ die Menge aller vollständigen Graphen. □

Wir erweitern die Signatur Σ_{GRAPH} um eine Funktion `clique`:

$$
\boxed{
\begin{array}{ll}
\textbf{signature} & \Sigma_{\mathrm{CLIQUE}} \\
\quad \textbf{import} & \Sigma_{\mathrm{GRAPH}}, \Sigma_{\mathrm{BOOL}} \\
\quad \textbf{functions} & \texttt{clique} : \texttt{Graph} \to \texttt{Bool} \\
\textbf{end signature} &
\end{array}
}
$$

und definieren die Σ_{CLIQUE}-Algebra **CLIQUE** durch

$$
\begin{array}{rcl}
\mathbf{CLIQUE}|_{\Sigma_{\mathrm{GRAPH}}} & = & \mathbf{GRAPH}, \\
\mathbf{CLIQUE}|_{\Sigma_{\mathrm{BOOL}}} & = & \mathbf{BOOL}
\end{array}
$$

und

$$
\texttt{clique}^{\mathbf{CLIQUE}}(G) \quad = \quad \begin{cases} \textit{true} & \text{, falls } G \in \mathcal{V} \\ \textit{false} & \text{, sonst} \end{cases}
$$

Das zentrale Ergebnis dieses Kapitels ist der folgende

Satz 4.3 (Spezifizierbarkeit von CLIQUE)
Der Datentyp **CLIQUE** *ist*

- *PL-spezifizierbar*
- *nicht QF-spezifizierbar.*

4.2 Prädikatenlogische Spezifizierbarkeit

In diesem Abschnitt wollen wir zeigen, daß **CLIQUE** *PL*-spezifizierbar ist. Hierzu geben wir eine Spezifikation CLIQUE an und zeigen, daß — bis auf Isomorphismus — deren einziges Modell gerade **CLIQUE** ist. Damit die Spezifikation lesbar bleibt, wollen wir folgende abkürzende Schreibweisen vereinbaren:[1]

$$
\begin{array}{rcl}
\textit{hasedge}(\mathtt{x}, \mathtt{y}, \mathtt{g}) & :\equiv & (\mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g}) = \mathtt{g}) \\
\textit{in}(\mathtt{x}, \mathtt{g}) & :\equiv & \exists \mathtt{y}.(\textit{hasedge}(\mathtt{x}, \mathtt{y}, \mathtt{g}) \vee \textit{hasedge}(\mathtt{y}, \mathtt{x}, \mathtt{g}))
\end{array}
$$

[1] Man beachte, daß dadurch keine neuen Funktionssymbole zur Signatur hinzugenommen werden. In der Spezifikation kann man sich jedes Auftreten der Abkürzung als textuell durch die Definition ersetzt vorstellen.

specification CLIQUE

import BOOL, NAT

sorts Graph

functions

$$
\begin{array}{lll}
\texttt{empty} & : & \to \texttt{Graph} \\
\texttt{connect} & : \texttt{Nat} \times \texttt{Nat} \times \texttt{Graph} & \to \texttt{Graph} \\
\texttt{clique} & : \texttt{Graph} & \to \texttt{Bool}
\end{array}
$$

variables

$$
\begin{array}{ll}
\texttt{x}, \texttt{y}, \texttt{u}, \texttt{v} & : \texttt{Nat} \\
\texttt{g}, \texttt{h} & : \texttt{Graph}
\end{array}
$$

axioms

$$
\begin{array}{ll}
(1) & \neg \mathit{hasedge}(\texttt{x}, \texttt{y}, \texttt{empty}) \\
(2) & \mathit{hasedge}(\texttt{x}, \texttt{y}, \texttt{connect}(\texttt{u}, \texttt{v}, \texttt{g})) \\
 & \quad \leftrightarrow (\texttt{x} = \texttt{u} \land \texttt{y} = \texttt{v}) \lor \mathit{hasedge}(\texttt{x}, \texttt{y}, \texttt{g}) \\
(3) & \texttt{g} = \texttt{h} \leftrightarrow \forall \texttt{x}, \texttt{y}.(\mathit{hasedge}(\texttt{x}, \texttt{y}, \texttt{g}) \leftrightarrow \mathit{hasedge}(\texttt{x}, \texttt{y}, \texttt{h})) \\
(4) & \texttt{clique}(\texttt{g}) = \texttt{true} \\
 & \quad \leftrightarrow \forall \texttt{x}, \texttt{y}.(\mathit{in}(\texttt{x}, \texttt{g}) \land \mathit{in}(\texttt{y}, \texttt{g}) \to \mathit{hasedge}(\texttt{x}, \texttt{y}, \texttt{g}))
\end{array}
$$

end specification

Dann ist unsere Behauptung:

Lemma 4.4 (*PL*-Spezifizierbarkeit von CLIQUE)
Für obige Spezifikation CLIQUE *gilt:*

$$\mathit{Mod}(\text{CLIQUE}) \quad = \quad [\textbf{CLIQUE}].$$

Beweis. Zum Beweis müssen wir zwei Aussagen zeigen, nämlich einerseits, daß **CLIQUE** $\in$ *Mod*(CLIQUE), und andererseits, daß die Spezifikation CLIQUE monomorph ist. Für den ersten Teil überzeugt man sich durch einfaches Nachrechnen, daß im Modell **CLIQUE** tatsächlich alle Axiome der Spezifikation zu wahr ausgewertet werden.

Für den Nachweis der Monomorphie verwenden wir Faktum 2.46 — wir müssen also zeigen, daß für jede Grundgleichung sie oder ihr Negat aus CLIQUE folgt. Hierzu unterscheiden wir nach den verschiedenen Typen von Grundgleichungen:

1. $\underline{\texttt{succ}^n(\texttt{zero}) = \texttt{succ}^m(\texttt{zero}) \text{ für } n, m \in \mathbb{N}}$

   Da NAT eine monomorphe Spezifikation ist, gilt für $m = n$, daß

   $$\text{CLIQUE} \quad \models \quad \texttt{succ}^n(\texttt{zero}) = \texttt{succ}^m(\texttt{zero}),$$

   und andernfalls

   $$\text{CLIQUE} \quad \models \quad \texttt{succ}^n(\texttt{zero}) \neq \texttt{succ}^m(\texttt{zero}).$$

2. $t_1^G = t_2^G$ mit Grundtermen t_1^G, t_2^G der Sorte `Graph`

   Am einfachsten ist dieser Fall unter Verwendung der oben eingeführten Abkürzungen einzusehen. Durch die Axiome (1) und (2) wird die Hilfsfunktion *hasedge* für alle Grundterme durch strukturelle Induktion festgelegt. In Axiom (3) wird die Gleichheit zweier beliebiger Graphen durch eine Äquivalenz definiert. Da die Ausdrücke auf der rechten Seite festgelegt sind, ist somit auch die Gleichheit zwischen zwei Graphen festgelegt.[2]

3. $t_1^B = t_2^B$ mit Grundtermen t_1^B, t_2^B der Sorte `Bool`

   Wir zeigen zunächst, daß für jeden Grundterm t^B der Sorte `Bool` entweder

   $$\begin{array}{rcl} \text{CLIQUE} & \models & t^B = \texttt{true} \\ \text{oder CLIQUE} & \models & t^B = \texttt{false}. \end{array}$$

   Für $t^B \in \{\texttt{true}, \texttt{false}\}$ ist die Behauptung klar. Für den anderen Fall betrachten wir zunächst Formeln $in(t^N, t^G)$ mit Grundtermen t^N, t^G. Da es sich dabei um eine Kurzschreibweise für eine Formel handelt, in der nur Grundgleichungen vorkommen, für die nach Punkt (2) die Gültigkeit durch CLIQUE festgelegt ist, gilt stets

   $$\begin{array}{rcl} \text{CLIQUE} & \models & in(t^N, t^G) \\ \text{oder CLIQUE} & \models & \neg in(t^N, t^G). \end{array}$$

   Nun können wir mit Hilfe von Axiom (4) folgern, daß für jeden Term t^G der Sorte `Graph` entweder

   $$\begin{array}{rcl} \text{CLIQUE} & \models & t^B = \texttt{true} \\ \text{oder CLIQUE} & \models & t^B \neq \texttt{true} \end{array}$$

   gilt, da auf der rechten Seite der Äquivalenz nach den vorangegangenen Ausführungen alle Grundgleichungen festgelegt sind. Mit Hilfe der Termerzeugtheit von `Bool` erhält man die Zwischenbehauptung.

   Die eigentliche Behauptung folgt nun sofort mit Hilfe der Termerzeugtheit von `Bool` und des Axioms $\texttt{true} \neq \texttt{false}$.

Durch die Spezifikation CLIQUE werden also alle Grundgleichungen festgelegt, die Spezifikation ist demnach monomorph.

■

[2] Zur Charakterisierung der Gleichheit könnte man auch ein Axiom ohne Verwendung von Quantoren angeben (vgl. Axiome (3)–(5) der Spezifikation CLIQUE^0 auf Seite 56). Allerdings wäre damit der Monomorphiebeweis deutlich aufwendiger.

4.3 Quantorenfreie Spezifizierbarkeit

In diesem Abschnitt wollen wir folgendes Lemma zeigen:

Lemma 4.5 (Nicht-$\boldsymbol{QF}$-Spezifizierbarkeit von CLIQUE)
Es gibt keine QF-Spezifikation SP mit $Mod(SP) = [\mathbf{CLIQUE}]$.

Zum Beweis geben wir zu jeder QF-Axiomatisierung ein termerzeugtes Modell $\mathbf{CLIQUE}_m$ an, das nicht zu **CLIQUE** isomorph ist. Diese Modelle $\mathbf{CLIQUE}_m$ unterscheiden sich von **CLIQUE** nur durch die Auswertung der Funktion `clique` an einer einzigen Stelle, nämlich für den vollständigen Graphen

$$G_m \quad := \quad \textstyle\bigcup_{i,j=0}^{m} \{(i,j)\}\,.$$

Genauer wählen wir

$$\begin{array}{lcl}
\mathbf{CLIQUE}_m|_{\Sigma_{\mathrm{GRAPH}}} & = & \mathbf{CLIQUE}|_{\Sigma_{\mathrm{GRAPH}}} \;=\; \mathbf{GRAPH}, \\
\mathbf{CLIQUE}_m|_{\Sigma_{\mathrm{BOOL}}} & = & \mathbf{CLIQUE}|_{\Sigma_{\mathrm{BOOL}}} \;=\; \mathbf{BOOL} \text{ und} \\
\texttt{clique}^{\mathbf{CLIQUE}_m}(G) & = & \begin{cases} \mathit{false} & \text{, für } G = G_m \\ \texttt{clique}^{\mathbf{CLIQUE}}(G) & \text{, sonst.} \end{cases}
\end{array}$$

Lemma 4.6 (Nicht-Standard-Modelle)
Ist SP eine QF-Spezifikation mit $\mathbf{CLIQUE} \in Mod(SP)$, so existiert ein $m \in \mathbb{N}$ mit $\mathbf{CLIQUE}_m \in Mod(SP)$.

Beweis von Lemma 4.5. Der Beweis von Lemma 4.5 ist damit einfach. Sei SP eine QF-Spezifikation. Ist $\mathbf{CLIQUE} \in Mod(SP)$, so folgt mit Lemma 4.6, daß für ein $m \in \mathbb{N}$ auch $\mathbf{CLIQUE}_m \in Mod(SP)$, also insbesondere $Mod(SP) \neq [\mathbf{CLIQUE}]$. Ist hingegen $\mathbf{CLIQUE} \notin Mod(SP)$, so folgt trivialerweise $Mod(SP) \neq [\mathbf{CLIQUE}]$. ∎

Da der Beweis von Lemma 4.6 ziemlich umfangreich und vor allem technisch aufwendig ist, wollen wir zunächst eine ausführliche Beschreibung der Beweisidee geben. Diese ist bewußt sehr informell und macht nur qualitative Aussagen. Eine Präzisierung erfolgt erst später, wobei wir bei den einzelnen Definitionen und Zwischenaussagen des Beweises stets auf die Bedeutung innerhalb des informellen Beweises verweisen wollen.

Die zugrundeliegende Idee bei der Wahl des Beispiels ist, daß die Vollständigkeit eines Graphen eine globale Eigenschaft ist: um sicherzustellen, daß ein Graph tatsächlich vollständig ist, muß eine Aussage über *alle* in ihm enthaltenen Kanten (bzw. Knoten) gemacht werden.

Dies kann jedoch — anschaulich gesprochen — von einer Formel ohne Quantoren nicht geleistet werden; eine solche Formel kann nur eine feste Zahl von Termen enthalten, und damit nur Relationen zwischen einer durch diese Zahl beschränkten Anzahl von Kanten ausdrücken. Eine induktive Definition ist auch nicht möglich, da der Abstand vollständiger Graphen — gemessen als Mächtigkeit ihrer symmetrischen Mengendifferenz — nicht beschränkt ist, sondern linear in der Größe der Graphen wächst. Ab einer bestimmten Größe der Graphen kann also eine Formel nicht mehr den Induktionsschritt ausdrücken.

Diese noch sehr informelle Argumentation läßt sich etwas präzisieren. Anders als im eigentlichen Beweis, in dem wir die folgenden etwas schwammigen Argumentationen vermeiden wollen und daher die Axiome als Disjunktionen von Literalen darstellen, stellen wir uns die Axiome hier als Implikationen geschrieben vor, so daß die Aussage über die Vollständigkeit ($\texttt{clique}(t^G) = \texttt{true}$) der Sukzedent der Implikation ist. Anhand syntaktischer Kriterien der Formel wählen wir den Graphen G_m „groß genug" für alle folgenden Aussagen.

Wenn sich tatsächlich die Vollständigkeit von G_m aus der Spezifikation folgern ließe ($\mathbf{CLIQUE}_m$ also kein Modell der Spezifikation wäre), müßte unter einer geeigneten Belegung in einem der Axiome für den Sukzedenten t^G zu G_m ausgewertet werden, und alle Vorbedingungen wären erfüllt.

Wäre eine Vorbedingung die Vollständigkeit eines „mit G_m verwandten" Graphen, d.h. eines Graphen, der durch eine Gleichung (bzw. mehrere Gleichungen mit „Zwischentermen"), mit G_m verbunden ist (das Axiom entspräche dann quasi einem Induktionsschritt), so könnten sich die beiden Graphen nicht in sehr vielen Kanten unterscheiden. Denn jede einzelne Gleichung kann ja zwischen den beiden beteiligten Graphen nur beschränkt viele Kanten ändern (hinzufügen bzw. entfernen).

Für ein genügend großes G_m können aber nicht beide Graphen gleichzeitig vollständig sein (der Abstand vollständiger Graphen wächst ja unbeschränkt), es sei denn, beide Graphen wären gleich. Dann wäre aber die Gültigkeit des Axioms unabhängig von der Auswertung von $\texttt{clique}(G_m)$, es ließe sich also nicht die Vollständigkeit von G_m folgern.

Enthält der Antezedent keine Vorbedingung, die die Vollständigkeit eines anderen mit G_m „verbundenen" Graphen besagt, so gibt es erst recht keine Anhaltspunkte, die Vollständigkeit von G_m zu schließen. Unter einer gegebenen Variablenbelegung „betrachtet" die Formel nur natürliche Zahlen bis zu einer bestimmten Schranke — was jenseits dieser Schranke passiert, verändert die Auswertung der Formel nicht.

So kann man zum Graphen G_m eine Kante zwischen genügend großen Zahlen hinzufügen, die von der Formel nicht bemerkt wird, wenn die Auswertung der anderen Variablen gleich bleibt. Damit weiterhin alle Vorbedingungen erfüllt bleiben, muß die Kante zu allen mit t^G „verbundenen"

Graphtermen hinzugefügt werden, da sonst die Gleichheiten zwischen diesen Termen verletzt würden. Daß die anderen Vorbedingungen erfüllt bleiben, erfordert keine weiteren Anstrengungen, so daß die Vollständigkeit eines unvollständigen Graphen folgt, nämlich die von G_m mit einer Extrakante.

Beide Fälle entsprechen nicht den Ansprüchen an eine korrekte und monomorphe Spezifikation der Vollständigkeit, so daß damit die Behauptung folgt.

Im nachfolgenden Beweis betrachten wir statt der (nicht eindeutigen) Form der Axiome als Implikation Axiome, die als Disjunktionen von Literalen geschrieben sind. Wenn wir die informell betrachteten Implikationen als Disjunktionen schreiben wollen, müssen wir alle Vorbedingungen negieren. Daher rühren im folgenden die etwas unnatürlich anmutenden Definitionen mit Hilfe von *Ungleichungen*.

Definition 4.7 (verbundene Terme)
Sei φ eine Disjunktion von Literalen. Zwei Terme t_1^G, t_2^G der Sorte `Graph` heißen **benachbart** in φ ($B_\varphi(t_1^G, t_2^G)$), wenn sie (gegebenenfalls als Unterterm) in φ vorkommen und t_1^G Unterterm von t_2^G ist oder es in φ ein Literal $t_1^G \neq t_2^G$ gibt.

Die transitive, symmetrische Hülle der Nachbarschaftsrelation B_φ wollen wir mit $\sim_\varphi$ bezeichnen.[3] Gilt $t_1^G \sim_\varphi t_2^G$, so heißen t_1^G und t_2^G **verbunden in** φ. □

Diese Definition formalisiert die schon oben informell eingeführte Nachbarschaft. Anschaulich bedeutet eine „Verbindung“ zwischen zwei Termen, daß die Auswertungen der Terme nicht allzu weit auseinander liegen können, zumindest, wenn die Ungleichungen nicht erfüllt sind. Das wird bei den Beweisen stets der Fall sein, wenn von $\sim_\varphi$ die Rede ist. In diesem Fall bedeutet die Existenz einer Verbindung auch, daß die verbundenen Terme nur gemeinsam geändert werden dürfen, wenn die Ungleichungen auch weiterhin nicht erfüllt sein sollen.

Als nächstes vereinbaren wir den oben erwähnten Abstand zweier Graphen als Mächtigkeit der symmetrischen Mengendifferenz, d.h. als Anzahl der Kanten, die man Hinzufügen oder Entfernen muß, um einen Graphen in den anderen zu überführen. Dieser semantischen Größe werden wir anschließend die syntaktische Größe connects(φ) einer Formel φ gegenüberstellen und Zusammenhänge als Lemmata nachweisen.

---

[3]Offenbar ist dann $\sim_\varphi$ eine Äquivalenzrelation auf der Menge der in φ vorkommenden Terme der Sorte `Graph`.

Definition 4.8 (Graphabstand)
Der **Abstand** $d(G_1, G_2)$ zweier Graphen $G_1, G_2 \subset \mathbb{N} \times \mathbb{N}$ ist die Mächtigkeit der symmetrischen Mengendifferenz

$$d(G_1, G_2) \;:=\; |(G_1 \setminus G_2) \cup (G_2 \setminus G_1)|\,.$$

□

Bemerkung 4.9 (zum Abstandsmaß von Graphen)
Durch das obige Abstandsmaß d ist auf der Menge der Graphen (bzw. allgemein jeder Menge von Mengen) eine Metrik erklärt. Denn man kann leicht nachprüfen, daß der Abstand tatsächlich folgende Bedingungen erfüllt:

1. *Definitheit*: $d(G_1, G_2) \geq 0$, und $d(G_1, G_2) = 0 \iff G_1 = G_2$.
2. *Symmetrie*: $d(G_1, G_2) = d(G_2, G_1)$.
3. *Dreiecks-Ungleichung*: $d(G_1, G_3) \leq d(G_1, G_2) + d(G_2, G_3)$.

Außerdem gilt für alle $G_1, G_2 \subseteq \mathbb{N}^2$, daß $d(G_1, G_2) \leq |G_1| + |G_2|$. □

Definition 4.10 (Anzahl von `connect`-Symbolen)
connects(ϕ) bzw. connects(t) bezeichnet die Anzahl der Vorkommen von `connect`-Symbolen in Formeln bzw. Termen und wird folgendermaßen induktiv definiert:

$$\begin{array}{lcl}
\mathrm{connects}(x) & := & 0, \text{ für Variablen } x \\
\mathrm{connects}(\texttt{true}) & := & 0 \\
\mathrm{connects}(\texttt{false}) & := & 0 \\
\mathrm{connects}(t^N) & := & 0 \\
\mathrm{connects}(\texttt{empty}) & := & 0 \\
\mathrm{connects}(\texttt{connect}(t_1^N, t_2^N, t^G)) & := & 1 + \mathrm{connects}(t^G) \\
\mathrm{connects}(\texttt{clique}(t^G)) & := & \mathrm{connects}(t^G) \\
 & & \\
\mathrm{connects}(\mathbf{false}) & := & 0 \\
\mathrm{connects}(t_1 = t_2) & := & \mathrm{connects}(t_1) + \mathrm{connects}(t_2) \\
\mathrm{connects}(\neg\varphi) & := & \mathrm{connects}(\varphi) \\
\mathrm{connects}(\varphi_1 \wedge \varphi_2) & := & \mathrm{connects}(\varphi_1) + \mathrm{connects}(\varphi_2).
\end{array}$$

□

Die Anzahl von `connect`-Symbolen in einer Formel ist das erwähnte Größenmaß. Anschaulich können zwischen zwei Termen durch jede Gleichung, die sie verbindet, nur so viele Kanten verändert werden (hinzugefügt oder entfernt) wie Funktionssymbole `connect` in der Gleichung auftreten. Insgesamt können sich also zwei verbundene Terme höchstens um connects(φ) Kanten unterscheiden, wenn die verbindenden Ungleichungen nicht erfüllt sind. Ein formaler Beweis dieser intuitiv klaren Aussage erfordert allerdings einigen Aufwand, ihm sind die folgenden beiden Lemmata gewidmet.

Lemma 4.11 (Kantendifferenz und connects)
*Sei s^G Unterterm von t^G und $v : X \to$ **GRAPH** eine Variablenbelegung. Dann gilt*

$$d\left((t^G)^{v,\mathbf{CLIQUE}}, (s^G)^{v,\mathbf{CLIQUE}}\right) \quad \leq \quad \mathrm{connects}(t^G) - \mathrm{connects}(s^G).$$

Beweis. Um die Formeln im weiteren Verlauf des Beweises lesbar zu halten, wollen wir kurz $G_t := (t^G)^{v,\mathbf{CLIQUE}}$, $G_s := (s^G)^{v,\mathbf{CLIQUE}}$ schreiben. Zum Beweis verwenden wir Induktion nach connects(t^G) − connects(s^G).

- Im Induktionsanfang ist connects(t^G) = connects(s^G), also $t^G = s^G$ und somit
$$d\,(G_t, G_s) \quad = \quad 0 \quad = \quad \mathrm{connects}(t^G) - \mathrm{connects}(s^G).$$
- Im Induktionsschritt ist
$$\mathrm{connects}(t^G) - \mathrm{connects}(s^G) \quad = \quad n+1 \quad > \quad 0,$$
also ist t^G von der Form $\texttt{connect}(t_1^N, t_2^N, t_1^G)$, connects($t_1^G$) $= n$, und s^G ist Unterterm von t_1^G. Für t_1^G ist demnach die Induktionsvoraussetzung anwendbar und liefert (mit $G_{t_1} := (t_1^G)^{v,\mathbf{CLIQUE}}$):
$$d\,(G_{t_1}, G_s) \quad \leq \quad \mathrm{connects}(t_1^G) - \mathrm{connects}(s^G).$$
Nach Definition von $\texttt{connect}^{\mathbf{GRAPH}}$ ist für $n_1 := (t_1^N)^{v,\mathbf{CLIQUE}}$ und $n_2 := (t_2^N)^{v,\mathbf{CLIQUE}}$
$$G_t \quad = \quad G_{t_1} \cup \{(n_1, n_2)\},$$
also
$$(G_t \setminus G_s) \cup (G_s \setminus G_t) \quad \subseteq \quad (G_{t_1} \setminus G_s) \cup (G_s \setminus G_{t_1}) \cup \{(n_1, n_2)\}.$$
Hieraus folgt
$$\begin{aligned} d\,(G_t, G_s) \quad &\leq \quad d\,(G_{t_1}, G_s) + |\{(n_1, n_2)\}| \\ &= \quad d\,(G_{t_1}, G_s) + 1 \\ &\overset{\text{I.V.}}{\leq} \quad \mathrm{connects}(t_1^G) - \mathrm{connects}(s^G) + 1 \\ &= \quad \mathrm{connects}(t^G) - \mathrm{connects}(s^G). \end{aligned}$$

■

Mit der Aussage, die wir nun für alle Paare von Termen und Untertermen zur Verfügung haben, können wir eine Abschätzung für den allgemeineren Fall von verbundenen Termen beweisen.

Lemma 4.12 (Kantendifferenz verbundener Terme)
Sei φ eine Disjunktion von Literalen und $v : X \to \mathbf{GRAPH}$ eine Variablenbelegung, so daß für alle Ungleichungen λ_j der Sorte `Graph` *in φ gilt: $\mathbf{CLIQUE}, v \not\models \lambda_j$. Sind t_a^G, t_e^G zwei beliebige, in φ verbundene Terme (die Indizes stehen hierbei für „Anfang“ und „Ende“ eines Pfades in der Nachbarschaftsrelation), so gilt:*

$$d\left((t_a^G)^{v,\mathbf{CLIQUE}}, (t_e^G)^{v,\mathbf{CLIQUE}}\right) \quad \leq \quad \mathrm{connects}(\varphi).$$

Beweis. Sind t_a^G, t_e^G in φ verbunden, so existiert nach Definition ein Pfad zwischen den beiden Termen in der Nachbarschaftsrelation. Die Länge eines minimalen solchen Pfades sei J. Der j-te Knoten des Pfades entspricht dabei einem Term s_j^G (mit $t_a^G = s_1^G, s_J^G = t_e^G$), die Kanten nach Definition der Nachbarschaft entweder einer Ungleichung $s_j^G \neq s_{j+1}^G$ in φ oder einer Untertermrelation $s_j^G \asymp s_{j+1}^G$. $\mathcal{J} \subseteq \{1, \ldots, J\}$ sei die Menge aller j, so daß die Kante zwischen s_j^G und s_{j+1}^G einer Untertermrelation entspricht.

Der Pfad kann keine zwei Untertermrelationen $s_i^G \asymp s_{i+1}^G$ und $s_j^G \asymp s_{j+1}^G$ mit $i < j$ enthalten, so daß $s_i^G, s_{i+1}^G, s_j^G, s_{j+1}^G$ Unterterme desselben Terms t^G sind. Denn träte dieser Fall ein, wäre entweder $s_i^G \preceq s_{j+1}^G$ oder $s_{j+1}^G \preceq s_i^G$. Dann könnte aber der Pfad um $j - i$ verkürzt werden, indem das Pfadstück $s_i^G, \ldots, s_{j+1}^G$ ersetzt würde durch eine einzige Kante $s_i^G \asymp s_{j+1}^G$, im Widerspruch zur Minimalität des Pfades (man überzeugt sich leicht, daß die obige Argumentation auch im Fall $j = i + 1$ gilt).

Nach Definition der Nachbarschaft existiert zu jedem $j \in \mathcal{J}$ ein Term t_j^G mit $s_j^G \preceq t_j^G$ und $s_{j+1}^G \preceq t_j^G$. Der vorangegangene Abschnitt stellt sicher, daß die t_j^G paarweise verschieden und auch keine Unterterme voneinander sind.

Für $j \notin \mathcal{J}$ gilt

$$d\left((s_j^G)^{v,\mathbf{CLIQUE}}, (s_{j+1}^G)^{v,\mathbf{CLIQUE}}\right) \quad = \quad 0,$$

da $\mathbf{CLIQUE}, v \not\models s_j^G \neq s_{j+1}^G$ nach Voraussetzung. Man erhält dann folgende Abschätzung

$$\begin{aligned} & d\left((t_a^G)^{v,\mathbf{CLIQUE}}, (t_e^G)^{v,\mathbf{CLIQUE}}\right) \\ \overset{\Delta\text{-Ungl.}}{\leq} \quad & \sum_{j=1}^{J-1} d\left((s_j^G)^{v,\mathbf{CLIQUE}}, (s_{j+1}^G)^{v,\mathbf{CLIQUE}}\right) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{j\in\mathcal{J}} d\left((s_j^G)^{v,\mathbf{CLIQUE}}, (s_{j+1}^G)^{v,\mathbf{CLIQUE}}\right) \\
&\overset{\text{Lemma 4.11}}{\leq} \sum_{j\in\mathcal{J}} \text{connects}(t_j^G) \\
&\leq \text{connects}(\varphi),
\end{aligned}
$$

und somit die Behauptung. ■

Nachdem wir nun eine Abschätzung für die Differenz der Kantenzahl von Graphen anhand syntaktischer Kriterien der Spezifikation hergeleitet haben, wollen wir als nächstes zeigen, daß der Abstand zwischen vollständigen Graphen unbeschränkt wächst, also jede beliebige Zahl überschreitet, wenn nur die Graphen groß genug gewählt werden. Zusammen mit der obigen Aussage ermöglicht uns dies, zu zeigen, daß eine induktive Spezifikation der Vollständigkeit nicht möglich ist.

Lemma 4.13 (Kantendifferenz vollständiger Graphen)
Sei $G \in \mathcal{V}$ ein vollständiger Graph, $m \in \mathbb{N}$ und $G \neq G_m$. Dann gilt

$$d\,(G, G_m) \;\geq\; 2m+1.$$

Beweis. Wir unterscheiden zwei Fälle:

1. $G_m \setminus G \neq \emptyset$, etwa $(x,y) \in G_m \setminus G$: Dann gilt $x \notin G$ oder $y \notin G$, da andernfalls wegen der Vollständigkeit von G auch $(x,y) \in G$, und somit $(x,y) \notin G_m \setminus G$ folgen würde. Sei o.B.d.A $x \notin G$. Für jedes $u = 0,\ldots,m$ ist $(x,u) \in G_m$ und $(x,u) \notin G$, genauso $(u,x) \in G_m$ und $(u,x) \notin G$. Also ist
$$\{(x,u),(u,x) \mid u = 0,\ldots,m\} \subseteq G_m \setminus G$$
und damit
$$
\begin{aligned}
d\,(G, G_m) &\geq |G_m \setminus G| \\
&\geq |\{(x,u),(u,x) \mid u = 0,\ldots,m\}| \\
&= 2m+1.
\end{aligned}
$$

2. $G_m \setminus G = \emptyset$. Da $G \neq G_m$ nach Voraussetzung, folgt $G \setminus G_m \neq \emptyset$, etwa $(x,y) \in G \setminus G_m$. Wie oben sei o.B.d.A. $x \notin G_m$. Es folgt hieraus sofort $x > m$. Da $G_m \subset G$, ist $u \in G$ für jedes $u = 0,\ldots,m$, und es folgt analog
$$\{(x,u),(u,x) \mid u = 0,\ldots,m\} \subseteq G \setminus G_m$$

und hieraus

$$\begin{aligned} d\,(G, G_m) &\geq |G \setminus G_m| \\ &\geq |\{(x,u),(u,x) \mid u = 0,\ldots,m\}| \\ &= 2m+2 \\ &\geq 2m+1. \end{aligned}$$

■

Durch die vorangegangenen Definitionen und Lemmata haben wir das Rüstzeug bereitgestellt, um den in der informellen Beschreibung des Beweises als erstes erwähnten Teil formell durchzuführen — also die Unmöglichkeit einer „induktiven“ Spezifikation. Im Fall, daß die Vollständigkeit eines Graphen ohne die Vollständigkeit eines verbundenen Graphen als Voraussetzung gefolgert werden soll, wenden wir die Technik an, dem Graphen eine Kante hinzuzufügen, die von der Formel „nicht bemerkt“ wird. Dazu müssen wir zunächst formal definieren, was wir darunter verstehen, daß eine Formel etwas „bemerkt“.

Definition 4.14 (Maximum einer Formel)
Sei $v : X \to \mathbf{GRAPH}$ eine Variablenbelegung. Das **Maximum** $\max_v$ von Formeln und Termen ist die größte natürliche Zahl, die in der Auswertung der Formel (des Terms) vorkommt, und zwar entweder als Auswertung eines Terms der Sorte `Nat` oder als Anfangs- oder Endpunkt einer Kante eines Terms der Sorte `Graph`. Eine induktive Definition dieser Größe lautet:

$$\begin{aligned} \max_v(t^N) &:= (t^N)^{v,\mathbf{CLIQUE}} \\ \max_v(\mathtt{true}) &:= 0 \\ \max_v(\mathtt{false}) &:= 0 \\ \max_v(t^G) &:= \max(x \in (t^G)^{v,\mathbf{CLIQUE}}) \\ \max_v(\mathtt{clique}(t^G)) &:= \max_v(t^G) \\ \\ \max_v(\mathbf{false}) &:= 0 \\ \max_v(t_1 = t_2) &:= \max(\max_v(t_1), \max_v(t_2)) \\ \max_v(\neg\varphi) &:= \max_v(\varphi) \\ \max_v(\varphi_1 \wedge \varphi_2) &:= \max(\max_v(\varphi_1), \max_v(\varphi_2)). \end{aligned}$$

□

Daß eine Formel eine Kante nicht erkennt, die zwischen Knoten verläuft, die außerhalb des „Sichtbereichs“ liegen, wird später zum großen Teil aus dem folgenden Lemma folgen.

Lemma 4.15 (nicht enthaltene Kante)
Für einen Term t^G der Sorte `Graph` *und $n \geq \max_v(t^G)$ gilt*

$$(n+1, n+2) \quad \notin \quad (t^G)^{v,\mathbf{CLIQUE}}.$$

Beweis. Der Beweis folgt unmittelbar aus der Definition von $\max_v(t^G)$. ■

Wir haben soeben ein Maß definiert, das uns erlaubt, Kanten zu Graphen hinzuzufügen und dabei zu erreichen, daß sie von der Formel nicht „bemerkt“ werden, also die Auswertung der Formel nicht beeinflussen. Als nächstes wollen wir eine formelle Definition der Menge der Graphen angeben, zu denen wir die Kanten hinzufügen, um die in der informellen Beschreibung geforderte Nichterfüllung der Ungleichungen zu garantieren.

Definition 4.16 (Urbilder von G_m)
Sei $v : X \to \mathbf{GRAPH}$ eine Variablenbelegung und $m \in \mathbb{N}$. Die Menge $U_{\varphi,v,m}$ der **Urbilder von** G_m **unter** v ist die Menge aller Terme t^G, die in der Formel φ vorkommen, und für die $(t^G)^{v,\mathbf{CLIQUE}} = G_m$.

Dann ist $[U_{\varphi,v,m}]_{\sim_\varphi}$ gerade die Menge der mit $U_{\varphi,v,m}$ unter φ verbundenen Terme, d.h. $t^G \in [U_{\varphi,v,m}]_{\sim_\varphi}$ genau dann, wenn ein $t_1^G \in U_{\varphi,v,m}$ existiert mit $t^G \sim_\varphi t_1^G$. □

Nachdem wir nun die Menge der Terme definiert haben, zu denen wir eine Kante hinzufügen wollen, geben wir als nächstes eine geeignete Variablenbelegung an und zeigen, daß diese auch das Gewünschte leistet.

Definition 4.17 (modifizierte Variablenbelegung)
Sei φ eine Disjunktion von Literalen, $v : X \to \mathbf{GRAPH}$ eine Variablenbelegung und $m, n \in \mathbb{N}$. Wir definieren eine modifizierte Variablenbelegung $v_{\varphi,m,n}$ durch

$$\begin{array}{lcl} v_{\varphi,m,n}(x^N) & := & v(x^N) \\ v_{\varphi,m,n}(x^B) & := & v(x^B) \\ v_{\varphi,m,n}(x^G) & := & \begin{cases} v(x^G) \cup \{(n+1, n+2)\} & \text{, falls } x^G \in [U_{\varphi,v,m}]_{\sim_\varphi} \\ v(x^G) & \text{, sonst.} \end{cases} \end{array}$$

□

Lemma 4.18 (Eigenschaften von $v_{\varphi,m,n}$)
Für jedes φ, m, n gelten folgende Aussagen:

1. $(t^N)^{v,\mathbf{CLIQUE}} = (t^N)^{v_{\varphi,m,n},\mathbf{CLIQUE}}$

2. *Ist $t^G \notin [U_{\varphi,v,m}]_{\sim_\varphi}$, so gilt*

$$(t^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} = (t^G)^{v,\mathbf{CLIQUE}}.$$

3. *Enthält t^G keine Variable x^G, so gilt*

$$(t^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} = (t^G)^{v,\mathbf{CLIQUE}}.$$

4. *Ist $t^G \in [U_{\varphi,v,m}]_{\sim_\varphi}$ und enthält t^G eine Variable x^G, so gilt*

$$(t^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} = (t^G)^{v,\mathbf{CLIQUE}} \cup \{(n+1, n+2)\}.$$

Beweis. Zum Beweis der Aussagen genügen einfache Induktionsbeweise über die Termtiefe von t^N bzw. t^G. ■

Das folgende Lemma ist der erste zentrale Bestandteil des eigentlichen Beweises. Es besagt, daß die Gültigkeit von Formeln auch in $\mathbf{CLIQUE}_m$ gesichert ist, wenn in der Auswertung gar keine Cliqueneigenschaft des Graphen G_m vorkommt. In der Schreibweise der anfänglichen Erläuterungen wird das Lemma vor allem dazu nützlich sein, den Fall zu behandeln, daß die Vorbedingung der Implikation (also in der Disjunktionsschreibweise also noch mindestens ein Literal außer der Cliqueneigenschaft von G_m) nicht erfüllt ist

Lemma 4.19 (cliquenfreie Literale in $\mathbf{CLIQUE}_m$)
*Sei $v : X \to$ **GRAPH** Variablenbelegung, $m \in \mathbb{N}$ und λ Literal, in dem kein Term* `clique`(t^G) *vorkommt, so daß $(t^G)^{v,\mathbf{CLIQUE}} = G_m$. Dann gilt*

$$\mathbf{CLIQUE}, v \models \lambda \implies \mathbf{CLIQUE}_m, v \models \lambda.$$

Beweis. Zum Beweis nehmen wir eine Fallunterscheidung für die verschiedenen Formen, die λ annehmen kann, vor:

- $\underline{\lambda \equiv (t_1^N = t_2^N) \text{ oder } \lambda \equiv (t_1^N \neq t_2^N)}$:

  Nach Definition gilt $\mathbf{CLIQUE}|_{\Sigma_{\mathrm{NAT}}} = \mathbf{CLIQUE}_m|_{\Sigma_{\mathrm{NAT}}}$. Damit erhalten wir für jeden Term t^N der Sorte `Nat`:

$$(t^N)^{v,\mathbf{CLIQUE}} = (t^N)^{v,\mathbf{CLIQUE}_m},$$

  woraus die Behauptung unmittelbar folgt.

- $\lambda \equiv (t_1^G = t_2^G)$ oder $\lambda \equiv (t_1^G \neq t_2^G)$:

  Mit Hilfe der Eigenschaft $\mathbf{CLIQUE}|_{\Sigma_{\mathrm{GRAPH}}} = \mathbf{CLIQUE}_m|_{\Sigma_{\mathrm{GRAPH}}}$ aus der Definition von **CLIQUE** zeigen wir, daß für jeden Term t^G der Sorte `Graph` gilt:

  $$(t^G)^{v,\mathbf{CLIQUE}} \quad = \quad (t^G)^{v,\mathbf{CLIQUE}_m}.$$

  Hieraus folgt die Behauptung dann unmittelbar.

- $\lambda \equiv (t_1^B = t_2^B)$ oder $\lambda \equiv (t_1^B \neq t_2^B)$:

  Wir zeigen, daß für alle Terme t^B der Sorte `Bool`, die nicht von den Voraussetzungen des Lemmas ausgeschlossen werden, gilt:

  $$(t^B)^{v,\mathbf{CLIQUE}} \quad = \quad (t^B)^{v,\mathbf{CLIQUE}_m}.$$

  Für $t^B \equiv$ `true` oder $t^B \equiv$ `false` folgt die Behauptung sofort aus der Definition von $\mathbf{CLIQUE}_m$. Für den Fall, daß $t^B = \texttt{clique}(t^G)$, verwenden wir die obige Gleichung

  $$(t^G)^{v,\mathbf{CLIQUE}} \quad = \quad (t^G)^{v,\mathbf{CLIQUE}_m}$$

  und die Eigenschaft von $\mathbf{CLIQUE}_m$, daß

  $$\texttt{clique}^{\mathbf{CLIQUE}}(G) \quad = \quad \texttt{clique}^{\mathbf{CLIQUE}_m}(G)$$

  für alle $G \neq G_m$ gilt. Für alle Fälle außer $(t^G)^{v,\mathbf{CLIQUE}} = G_m$ ist die Aussage damit gezeigt, und dieser Fall tritt laut Voraussetzung des Lemmas ja nicht auf.

■

Um, wie in der Einleitung zum Beweis angedeutet, dem Graphen G_m eine Kante hinzuzufügen, ohne daß sich dadurch die Auswertung der Formel verändert, haben wir die Kante mit Hilfe der Belegung $v_{\varphi,m,n}$ zu allen verbundenen Graphen hinzugefügt. Im nachfolgenden Lemma wollen wir zeigen, daß dadurch keine Literale plötzlich zu *true* ausgewertet werden, außer solchen, die Aussagen über die Vollständigkeit von mit G_m verbundenen Graphen machen. Um die Formulierung nicht zu umständlich zu machen, folgt zunächst noch eine

Definition 4.20 (U-Cliquenliteral)
Sei φ Disjunktion von Literalen, $m \in \mathbb{N}$ und $v : X \to \mathbf{GRAPH}$ eine Variablenbelegung. Ein $[U_{\varphi,v,m}]_{\sim_\varphi}$**-Cliquenliteral** ist ein Literal, das einen Term der Form $\texttt{clique}(t^G)$ mit $t^G \in [U_{\varphi,v,m}]_{\sim_\varphi}$ enthält. □

Lemma 4.21 (cliquenfreie Literale)
Sei $\varphi = \bigvee_{j=1}^{J} \lambda_j$, $v : X \to \mathbf{GRAPH}$ Variablenbelegung, $m \geq \mathrm{connects}(\varphi)$ und $n \geq \max_v(\varphi)$. Weiterhin gelte für alle $j \in \{1, \ldots, J\}$: Ist λ_j kein $[U_{\varphi,v,m}]_{\sim_\varphi}$-Cliquenliteral, so $\mathbf{CLIQUE}, v \not\models \lambda_j$.

Dann gilt auch für alle $j \in \{1, \ldots, J\}$, die keine $[U_{\varphi,v,m}]_{\sim_\varphi}$-Cliquenliterale sind, daß

$$\mathbf{CLIQUE}, v_{\varphi,m,n} \quad \not\models \quad \lambda_j$$

Beweis. Wir wählen ein beliebiges der λ_j, das kein $[U_{\varphi,v,m}]_{\sim_\varphi}$-Cliquenliteral ist (im folgenden fest), und unterscheiden die Formen, die λ_j haben kann:

- $\underline{\lambda_j \equiv (t_1^N = t_2^N) \text{ oder } \lambda_j \equiv (t_1^N \neq t_2^N)}$:

  Die Behauptung folgt unmittelbar aus Lemma 4.18.

- $\underline{\lambda_j \equiv (t_1^G = t_2^G) \text{ oder } \lambda_j \equiv (t_1^G \neq t_2^G)}$:

  Ist $t^G \notin [U_{\varphi,v,m}]_{\sim_\varphi}$, so folgt mit Lemma 4.18, daß

  $$(t^G)^{v,\mathbf{CLIQUE}} \quad = \quad (t^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}},$$

  woraus die Behauptung sofort für $t_1^G, t_2^G \notin [U_{\varphi,v,m}]_{\sim_\varphi}$ folgt.

  Ist entweder t_1^G oder t_2^G in $[U_{\varphi,v,m}]_{\sim_\varphi}$ (o.B.d.A. sei dies t_1^G), so muß λ_j eine Gleichung $t_1^G = t_2^G$ sein, da andernfalls nach Definition 4.16 folgen würde, daß $t_2^G \in [U_{\varphi,v,m}]_{\sim_\varphi}$. Nach Lemma 4.18 und Lemma 4.15 ist dann

  $$\begin{array}{rcl} (n+1, n+2) & \in & (t_1^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} \text{ und} \\ (n+1, n+2) & \notin & (t_2^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}}. \end{array}$$

  Hiermit folgt, wie behauptet, $\mathbf{CLIQUE}, v_{\varphi,m,n} \not\models \lambda_j$.

  Sind $t_1^G, t_2^G \in [U_{\varphi,v,m}]_{\sim_\varphi}$, so unterscheiden wir erneut drei Fälle:

  1. Enthalten weder t_1^G noch t_2^G eine Variable x^G der Sorte `Graph`, so verwenden wir die Aussage $(t^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} = (t^G)^{v,\mathbf{CLIQUE}}$ aus Lemma 4.18, womit sofort die Behauptung folgt.

2. Enthalten sowohl t_1^G als auch t_2^G Variablen x_1^G, x_2^G (nicht notwendig verschieden) der Sorte `Graph`, so folgt (ebenfalls mit Lemma 4.18), daß

$$\begin{aligned}(t_1^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} &= (t_1^G)^{v,\mathbf{CLIQUE}} \cup (n+1, n+2) \text{ und}\\ (t_2^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} &= (t_2^G)^{v,\mathbf{CLIQUE}} \cup (n+1, n+2).\end{aligned}$$

Da $n \geq \max_v(\varphi)$, ist nach Lemma 4.15

$$\begin{aligned}(n+1, n+2) &\notin (t_1^G)^{v,\mathbf{CLIQUE}} \text{ und}\\ (n+1, n+2) &\notin (t_2^G)^{v,\mathbf{CLIQUE}}.\end{aligned}$$

Also folgt

$$\begin{aligned}&(t_1^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} = (t_2^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}}\\ \iff \quad &(t_1^G)^{v,\mathbf{CLIQUE}} = (t_2^G)^{v,\mathbf{CLIQUE}},\end{aligned}$$

und hieraus die Behauptung.

3. Enthält genau ein Term eine Variable x^G der Sorte `Graph` (o.B.d.A sei dies t_1^G), so enthält der andere Term (t_2^G) die Konstante `empty`. Dann ist `empty` $\in [U_{\varphi,v,m}]_{\sim_\varphi}$. Es existiert also ein Term t^G mit $(t^G)^{v,\mathbf{CLIQUE}} = G_m$ und `empty` $\sim_\varphi t^G$. Mit Lemma 4.12 (dessen Voraussetzungen alle erfüllt sind — an dieser Stelle benötigen wir die starke Voraussetzung des Nicht-Erfülltseins aller Literale außer den $[U_{\varphi,v,m}]_{\sim_\varphi}$-Cliquenliteralen) folgt

$$\begin{aligned}(m+1)^2 &= \left|(t^G)^{v,\mathbf{CLIQUE}}\right|\\ &= d\left((t^G)^{v,\mathbf{CLIQUE}}, \emptyset\right)\\ &\leq \text{connects}(\varphi)\\ &\leq m,\end{aligned}$$

also ein Widerspruch, der dritte Fall kann in Wirklichkeit gar nicht auftreten.

- $\underline{\lambda_j \equiv (t_1^B = t_2^B) \text{ oder } \lambda_j \equiv (t_1^B \neq t_2^B)\text{:}}$

  Wir zeigen für jeden Term t^B der Sorte `Bool`, der den Voraussetzungen des Lemmas genügt, daß

$$(t^B)^{v,\mathbf{CLIQUE}} = (t^B)^{v_{\varphi,m,n},\mathbf{CLIQUE}}.$$

Für die Terme `true` und `false` ist die Behauptung offensichtlich. Andernfalls ist t^B von der Form $\texttt{clique}(t^G)$, und nach Voraussetzung ist $t^G \notin [U_{\varphi,v,m}]_{\sim_\varphi}$. Mit Lemma 4.18 folgt dann

$$(t^G)^{v,\mathbf{CLIQUE}} \;=\; (t^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}}$$

und hieraus die Behauptung.

■

Als nächstes folgt das zweite zentrale Lemma in unserer Argumentation. Wir setzen die Gültigkeit eines Axioms in Disjunktionsform (entsprechend also auch als Implikation) unter einer Variablenbelegung und ihrer oben definierten Modifikation in **CLIQUE** voraus und folgern daraus die Gültigkeit unter derselben Belegung in $\mathbf{CLIQUE}_m$. Die Gültigkeit unter der modifizierten Variablenbelegung entspricht dabei dem „Hinzufügen“ der Kante wie in der Beschreibung des Beweises. Daß die Formel auch unter dieser Belegung gültig ist, stellt sicher, daß sie nicht von der zweiten Bauart (nicht-induktiv) sein kann. Damit bleibt noch der induktive Fall, und in diesem Fall wird gezeigt, daß die Voraussetzung unter $\mathbf{CLIQUE}_m$ zu *false* ausgewertet wird.

Lemma 4.22 (Gültigkeit von Disjunktionen in $\mathbf{CLIQUE}_m$)
Seien $\varphi = \bigvee_{j=1}^{J} \lambda_j$ eine Disjunktion von Literalen, $v : X \to \mathbf{GRAPH}$ Variablenbelegung, $m \geq \mathrm{connects}(\varphi)$ und $n \geq \max_v(\varphi)$. Gelten

$$\begin{array}{lcl} \mathbf{CLIQUE}, v & \models & \varphi \text{ und} \\ \mathbf{CLIQUE}, v_{\varphi,m,n} & \models & \varphi, \end{array}$$

so folgt

$$\mathbf{CLIQUE}_m, v \models \varphi.$$

Beweis. Enthält φ mindestens ein Literal λ mit $\mathbf{CLIQUE}, v \models \lambda$, in dem kein Term $\texttt{clique}(t^G)$ mit $(t^G)^{v,\mathbf{CLIQUE}} = G_m$ vorkommt, so folgt mit Lemma 4.19, daß $\mathbf{CLIQUE}_m, v \models \lambda$. Also gilt auch $\mathbf{CLIQUE}_m, v \models \varphi$.

Andernfalls enthalten alle Literale λ mit $\mathbf{CLIQUE}, v \models \lambda$ einen Term $\texttt{clique}(t^G)$, so daß $(t^G)^{v,\mathbf{CLIQUE}} = G_m$. Da $\mathbf{CLIQUE}, v \models \varphi$, muß auch mindestens ein solches Literal λ_j mit einem Term $\texttt{clique}(t_j^G)$ existieren.

Aus Lemma 4.21 folgt, daß $[U_{\varphi,v,m}]_{\sim_\varphi}$-Cliquenliterale λ die einzigen sein können, für die $\mathbf{CLIQUE}, v_{\varphi,m,n} \models \lambda$ gelten kann, und da nach Voraussetzung $\mathbf{CLIQUE}, v_{\varphi,m,n} \models \varphi$, muß auch mindestens ein solches Literal λ_i mit einem Term $t_i^G \in [U_{\varphi,v,m}]_{\sim_\varphi}$ existieren. Da für alle Terme t^G, die eine Variable $x^G \in [U_{\varphi,v,m}]_{\sim_\varphi}$ enthalten, nach Lemma 4.18 und 4.15 gilt, daß

$$\begin{array}{lcl} (n+1, n+2) & \in & (t^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} \text{ und} \\ (n+1, n+1) & \notin & (t^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}}, \end{array}$$

ist

$$\texttt{clique}^{\mathbf{CLIQUE}}((t_i^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}}) \quad = \quad \mathit{false}.$$

Also muß λ_i von einer der folgenden beiden Formen sein:

1. $\texttt{clique}(t_i^G) = t^B$, mit $(t^B)^{v_{\varphi,m,n},\mathbf{CLIQUE}} = \mathit{false}$.

2. $\texttt{clique}(t_i^G) \neq t^B$, mit $(t^B)^{v_{\varphi,m,n},\mathbf{CLIQUE}} = \mathit{true}$.

Wir zeigen nun

$$\begin{array}{lcl} \texttt{clique}^{\mathbf{CLIQUE}_m}((t_i^G)^{v,\mathbf{CLIQUE}}) & = & \mathit{false} \text{ und} \\ (t^B)^{v,\mathbf{CLIQUE}_m} & = & (t^B)^{v_{\varphi,m,n},\mathbf{CLIQUE}}. \end{array}$$

Für den Nachweis der ersten Aussage unterscheiden wir zwei Fälle, je nach Auswertung von t_i^G (wie schon in der Bemerkung zum Lemma angedeutet). Ist

$$(t_i^G)^{v,\mathbf{CLIQUE}_m} \quad = \quad (t_i^G)^{v,\mathbf{CLIQUE}} \quad = \quad G_m,$$

so folgt die Aussage wegen $\texttt{clique}^{\mathbf{CLIQUE}_m}(G_m) = \mathit{false}$. Andernfalls ist $(t_i^G)^{v,\mathbf{CLIQUE}} \neq G_m$, aber t_i^G und t_j^G sind in φ unter v verbunden. Nach Lemma 4.12 gilt dann

$$d\left((t_j^G)^{v,\mathbf{CLIQUE}}, G_m\right) \quad \leq \quad \text{connects}(\varphi) \quad \leq \quad m,$$

und Lemma 4.13 impliziert $\texttt{clique}^{\mathbf{CLIQUE}_m}((t_i^G)^{v,\mathbf{CLIQUE}_m}) = \mathit{false}$.

Für die zweite Aussage unterscheiden wir wiederum die verschiedenen Formen, die t^B haben kann. Ist t^B nicht von der Form $\texttt{clique}(t^G)$, so daß t^G eine Variable $x^G \in [U_{\varphi,v,m}]_{\sim_\varphi}$ enthält, so ist die Behauptung leicht einzusehen, da die Auswertung sich nach Definition von $v_{\varphi,m,n}$ und $\mathbf{CLIQUE}_m$ nicht unterscheidet (vgl. Beweise zu Lemmata 4.19 und 4.21). Andernfalls kann es sich nur um den ersten Fall handeln, da nach Definition von $v_{\varphi,m,n}$ $(t^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}}$ nicht vollständig sein kann. Dann läßt sich die obige Begründung der ersten Aussage genauso wie für t_i^G auch für t^G durchführen und liefert die Behauptung.

Beide Aussagen zusammen ergeben, daß

$$\mathbf{CLIQUE}_m, v \quad \models \quad \lambda_i.$$

Somit folgt schließlich

$$\mathbf{CLIQUE}_m, v \quad \models \quad \varphi.$$

■

Nachdem wir ein entsprechendes Lemma für einzelne Axiome als Disjunktionen bereitgestellt haben, bleibt nun noch die einfache Anwendung auf die Axiome einer Spezifikation in konjunktiver Normalform.

Lemma 4.23 (Gültigkeit in $\mathbf{CLIQUE}_m$)
Sei $\phi = \bigwedge_{k=1}^{K} \varphi_k$ mit $\varphi_k = \bigvee_{j=1}^{J_k} \lambda_{k,j}$ eine Formel in konjunktiver Normalform, und $m \geq \mathrm{connects}(\phi)$. Dann gilt

$$\mathbf{CLIQUE} \models \phi \implies \mathbf{CLIQUE}_m \models \phi$$

Beweis. Wir zeigen für jedes φ_k $(k = 1, \ldots, K)$, daß

$$\mathbf{CLIQUE} \models \varphi_k \implies \mathbf{CLIQUE}_m \models \varphi_k.$$

Hieraus folgt dann unmittelbar die gewünschte Aussage.

Sei also ein $k \in \{1, \ldots, K\}$ fest gewählt, und es gelte

$$\mathbf{CLIQUE} \models \varphi_k.$$

Ist $v : X \to \mathbf{GRAPH}$ eine beliebige Variablenbelegung, so gelten

$$\begin{array}{lcl} \mathbf{CLIQUE}, v & \models & \varphi_k \text{ und} \\ \mathbf{CLIQUE}, v_{\varphi,m,n} & \models & \varphi_k. \end{array}$$

Mit Lemma 4.22 folgt

$$\mathbf{CLIQUE}_m, v \models \varphi_k.$$

Da v beliebig gewählt war, folgt

$$\mathbf{CLIQUE}_m \models \varphi_k.$$

■

Damit können wir nun endlich den Beweis der Behauptung dieses Abschnitts angehen. Durch das starke Hilfsmittel ist der Beweis ziemlich naheliegend.

Beweis von Lemma 4.6. Sei SP eine beliebige QF-Spezifikation über der Signatur Σ_{CLIQUE}. Wir bilden die Konjunktion aller Axiome und bringen die resultierende Formel in konjunktive Normalform. Das Ergebnis sei mit ϕ bezeichnet. Mit $m \geq \mathrm{connects}(\phi)$ sind dann die Voraussetzungen von Lemma 4.23 erfüllt, und man erhält

$$\begin{array}{rcl} \mathbf{CLIQUE} \in Mod(SP) & \implies & \mathbf{CLIQUE} \models \phi \\ & \implies & \mathbf{CLIQUE}_m \models \phi \\ & \implies & \mathbf{CLIQUE}_m \in Mod(SP). \end{array}$$

■

Kapitel 5

Eine Quantorenhierarchie

Im vorigen Kapitel haben wir anhand des Beispiels **CLIQUE** gezeigt, daß es Datentypen gibt, zu deren Spezifikation Quantoren vonnöten sind, d.h. wir haben gezeigt, daß mit voller Prädikatenlogik eine monomorphe Spezifikation von **CLIQUE** möglich ist, nicht jedoch mit quantorenfreien Axiomen.

In der Spezifikation CLIQUE waren wir mit Quantoren relativ großzügig — beispielsweise haben wir schon in Fußnote 2 auf Seite 36 erwähnt, daß man auf mindestens einen Quantor hätte verzichten können. Es stellt sich die Frage, wie viele Quantoren denn nun tatsächlich für die Spezifikation von **CLIQUE** nötig sind. Hat man dann eine Mindestanzahl von Quantoren für die Spezifikation bestimmt und nachgewiesen, so könnte man sich weiterhin fragen, ob es denn auch Datentypen gibt, die mit weniger Quantoren monomorph spezifizierbar sind, nicht jedoch ohne Quantoren, bzw. Datentypen, zu deren Spezifikation man noch mehr Quantoren braucht als zur Spezifikation von **CLIQUE**.

Allgemein lautet die Fragestellung: Gibt es zu jedem $k \in \mathbb{N}$ einen Datentyp A_k, so daß A_k zwar mit $k+1$, nicht jedoch mit k Quantoren spezifizierbar ist? Mit anderen Worten: Bilden die Mengen der mit $k \in \mathbb{N}$ Quantoren spezifizierbaren Datentypen bezüglich k eine Hierarchie? Wir werden in diesem Kapitel diese Frage positiv beantworten, indem wir für jedes k einen geeigneten Datentypen A_k angeben. Auch für dieses Kapitel wollen wir den Bereich des Berechenbaren nicht verlassen und insbesondere unter einem Datentyp stets einen *berechenbaren* Datentyp verstehen.

Vorher müssen wir jedoch einige Begriffe klären, insbesondere, was wir unter der Anzahl von Quantoren in einer Spezifikation verstehen wollen. Wir haben bisher ja bei Spezifikationen die Äquivalenz $\varphi \leftrightarrow \psi$ als abkürzende Schreibweise für $(\varphi \to \psi) \wedge (\psi \to \varphi)$ verwendet. Enthalten φ oder ψ Quantoren, so hätten wir offenbar bei Verzicht auf die Abkürzung $\leftrightarrow$ mehr Quantoren in unserer Spezifikation als unter Verwendung von $\leftrightarrow$. Daher wollen wir vereinbaren, die Anzahl von Quantoren nur anhand *echter* Formeln ohne Abkürzungen zu messen. Damit können wir jetzt definieren:

Definition 5.1 (Quantorenzahl und Hierarchieklassen)
Die **Quantorenanzahl** quants(SP) einer Spezifikation SP ist die Anzahl der Auftreten von '$\forall$' und '$\exists$' in den Axiomen von SP.

Quan_k bezeichnet die Menge aller berechenbaren Datentypen, für die eine monomorphe Spezifikation SP mit $\mathrm{quants}(SP) \leq k$ existiert.[1] □

Faktum 5.2 (Eigenschaften der $\mathbf{Quan}_k$)
Die nachfolgenden Eigenschaften der Mengen Quan_n *folgen unmittelbar aus der Definition:*

1. *Für jedes* $k \in \mathbb{N}$ *gilt* $\mathrm{Quan}_k \subseteq \mathrm{Quan}_{k+1}$.
2. $\mathrm{Quan}_0 = \{A \mid AQF\text{-}\textit{spezifizierbar}\}$.
3. $\bigcup_{k=0}^{\infty} \mathrm{Quan}_k = \{A \mid APL\text{-}\textit{spezifizierbar}\}$.

Ziel dieses Kapitels wird es sein, zu zeigen, daß die Inklusion in Aussage (1) von Faktum 5.2 echt ist. Die Behauptung lautet also:

Satz 5.3 (Hierarchie der $\mathbf{Quan}_k$)
Die Quan_k *bilden eine echte Hierarchie, d.h. für jedes* $k \in \mathbb{N}$ *gilt:*

$$\mathrm{Quan}_{k+1} \setminus \mathrm{Quan}_k \quad \neq \quad \emptyset.$$

Zum Beweis werden wir das in Kapitel 4 vorgestellte Beispiel modifizieren, so daß der durch die Modifikation entstehende Datentyp jeweils ein Vertreter der uns gerade interessierenden Klasse ist. Dabei dient der Datentyp, den wir zum Beweis der Behauptung für den Fall $k = 0$ konstruieren, als Grundlage für die Fälle $k > 0$, so daß wir ihn zunächst getrennt untersuchen.

5.1 Spezifikation mit einem Quantor

Zunächst wollen wir Satz 5.3 für den Fall $k = 0$ nachweisen. Betrachten wir die Spezifikation CLIQUE auf Seite 35 und lösen die darin enthaltenen Äquivalenzen und Abkürzungen auf, so erhalten wir zunächst eine Spezifikation mit 10 Quantoren. Auch durch größere Änderungen an der Spezifikation ist

[1] Dies impliziert auch, daß eine monomorphe Spezifikation SP mit $\mathrm{quants}(SP) = k$ existiert, da man ja die „noch fehlenden“ Quantoren dadurch hinzufügen kann, daß man eine Tautologie hinzunimmt und die benötigte Anzahl von Quantoren davorschreibt.

nicht offensichtlich, wie die Zahl der Quantoren auf weniger als 4 reduziert werden soll.[2]

Daher modifizieren wir den Datentyp **CLIQUE** zu einem neuen Datentyp $\mathbf{CLIQUE}^0$, um tatsächlich die Spezifizierbarkeit mit einem Quantor sicherzustellen. Wir vereinbaren die Signatur

signature $\Sigma_{\mathrm{CLIQUE}^0}$				
import	Σ_{CLIQUE}			
sorts	`Nat-Pair`			
functions	`in`	: `Nat` $\times$ `Graph`	$\to$	`Bool`
	`pair`	: `Nat` $\times$ `Nat`	$\to$	`Nat-Pair`
	`first`	: `Nat-Pair`	$\to$	`Nat`
	`second`	: `Nat-Pair`	$\to$	`Nat`
end signature				

und dazu die $\Sigma_{\mathrm{CLIQUE}^0}$-Algebra $\mathbf{CLIQUE}^0$ durch

$$\begin{array}{lcl}
\mathbf{CLIQUE}^0|_{\Sigma_{\mathrm{CLIQUE}}} & = & \mathbf{CLIQUE} \\
\mathtt{Nat\text{-}Pair}^{\mathbf{CLIQUE}^0} & = & \mathbb{N} \times \mathbb{N} \\
\mathtt{in}^{\mathbf{CLIQUE}^0}(x, G) & = & \begin{cases} \mathit{true} & \text{, falls } x \in G \\ \mathit{false} & \text{, falls } x \notin G \end{cases} \\
\mathtt{pair}^{\mathbf{CLIQUE}^0}(x, y) & = & (x, y) \\
\mathtt{first}^{\mathbf{CLIQUE}^0}((x, y)) & = & x \\
\mathtt{second}^{\mathbf{CLIQUE}^0}((x, y)) & = & y.
\end{array}$$

Dann lautet unsere Behauptung:

Lemma 5.4 (Spezifizierbarkeit von $\mathbf{CLIQUE}^0$)
$\mathbf{CLIQUE}^0 \in \mathrm{Quan}_1 \setminus \mathrm{Quan}_0$.

Wir werden in getrennten Abschnitten die Teilaussagen $\mathbf{CLIQUE}^0 \in \mathrm{Quan}_1$ und $\mathbf{CLIQUE}^0 \notin \mathrm{Quan}_0$ nachweisen.

5.1.1 $\mathbf{CLIQUE}^0 \in \mathbf{Quan}_1$

Um zu zeigen, daß $\mathbf{CLIQUE}^0 \in \mathrm{Quan}_0$, geben wir einfach eine geeignete Spezifikation CLIQUE^0 an und weisen nach, daß diese den Datentypen monomorph spezifiziert. Wie schon in Kapitel 4 vereinbaren wir die Abkürzung

$$\mathit{hasedge}(\mathtt{x}, \mathtt{y}, \mathtt{g}) \quad :\equiv \quad (\mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g}) = \mathtt{g}).$$

[2] In dieser Arbeit wird nicht geklärt, welches die kleinste Anzahl von Quantoren ist, mit der **CLIQUE** monomorph spezifiziert werden kann.

Damit sieht unsere Spezifikation CLIQUE^0 dann wie folgt aus:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textbf{specification}\ \ \mathrm{CLIQUE}^0} \\
\quad\textbf{import} & \mathrm{BOOL}, \mathrm{NAT} \\
\quad\textbf{sorts} & \texttt{Nat-Pair}, \texttt{Graph} \\
\quad\textbf{functions} & \begin{array}{lll}
\texttt{empty} & : & \to \texttt{Graph} \\
\texttt{connect} & : \texttt{Nat} \times \texttt{Nat} \times \texttt{Graph} & \to \texttt{Graph} \\
\texttt{in} & : \texttt{Nat} \times \texttt{Graph} & \to \texttt{Bool} \\
\texttt{clique} & : \texttt{Graph} & \to \texttt{Bool} \\
\texttt{pair} & : \texttt{Nat} \times \texttt{Nat} & \to \texttt{Nat-Pair} \\
\texttt{first} & : \texttt{Nat-Pair} & \to \texttt{Nat} \\
\texttt{second} & : \texttt{Nat-Pair} & \to \texttt{Nat}
\end{array} \\
\quad\textbf{variables} & \begin{array}{ll}
\texttt{x}, \texttt{y}, \texttt{u}, \texttt{v} & : \texttt{Nat} \\
\texttt{g}, \texttt{h} & : \texttt{Graph} \\
\texttt{p} & : \texttt{Nat-Pair}
\end{array} \\
\quad\textbf{axioms} & \begin{array}{ll}
(1) & \texttt{first}(\texttt{pair}(\texttt{x}, \texttt{y})) = \texttt{x} \\
(2) & \texttt{second}(\texttt{pair}(\texttt{x}, \texttt{y})) = \texttt{y} \\
(3) & \neg \mathit{hasedge}(\texttt{x}, \texttt{y}, \texttt{empty}) \\
(4) & \mathit{hasedge}(\texttt{x}, \texttt{y}, \texttt{connect}(\texttt{u}, \texttt{v}, \texttt{g})) \\
 & \qquad \leftrightarrow (\texttt{x} = \texttt{u} \land \texttt{y} = \texttt{v}) \lor \mathit{hasedge}(\texttt{x}, \texttt{y}, \texttt{g}) \\
(5) & \texttt{connect}(\texttt{x}, \texttt{y}, \texttt{connect}(\texttt{u}, \texttt{v}, \texttt{g})) \\
 & \qquad = \texttt{connect}(\texttt{u}, \texttt{v}, \texttt{connect}(\texttt{x}, \texttt{y}, \texttt{g})) \\
(6) & \texttt{in}(\texttt{x}, \texttt{empty}) = \texttt{false} \\
(7) & \texttt{in}(\texttt{x}, \texttt{connect}(\texttt{u}, \texttt{v}, \texttt{g})) = \texttt{true} \\
 & \qquad \leftrightarrow \texttt{in}(\texttt{x}, \texttt{g}) = \texttt{true} \lor \texttt{x} = \texttt{u} \lor \texttt{x} = \texttt{v} \\
(8) & (\texttt{in}(\texttt{first}(\texttt{p}), \texttt{g}) \land \texttt{in}(\texttt{second}(\texttt{p}), \texttt{g}) \\
 & \qquad\quad \land \neg \mathit{hasedge}(\texttt{first}(\texttt{p}), \texttt{second}(\texttt{p}), \texttt{g})) \\
 & \qquad \to \texttt{clique}(\texttt{g}) = \texttt{false} \\
(9) & (\forall \texttt{p}.(\texttt{in}(\texttt{first}(\texttt{p}), \texttt{g}) \land \texttt{in}(\texttt{second}(\texttt{p}), \texttt{g}) \\
 & \qquad\quad \to \mathit{hasedge}(\texttt{first}(\texttt{p}), \texttt{second}(\texttt{p}), \texttt{g}))) \\
 & \qquad \to \texttt{clique}(\texttt{g}) = \texttt{true}
\end{array} \\
\multicolumn{2}{l}{\textbf{end specification}}
\end{array}
$$

Offensichtlich kommt in der Spezifikation CLIQUE^0 genau ein Quantor vor, so daß wir zum Nachweis unserer Behauptung noch die folgende Aussage zeigen müssen:

Lemma 5.5 (CLIQUE^0 spezifiziert CLIQUE^0)
$\mathit{Mod}(\mathrm{CLIQUE}^0) = [\mathbf{CLIQUE}^0]$.

Beweis. Der Beweis des Lemmas ist vergleichbar mit dem von Lemma 4.4, allerdings gestaltet er sich durch die größere Anzahl von Sorten und Funktionen und die Vermeidung von Äquivalenzen aufwendiger.

Wiederum kann man einfach nachprüfen, daß tatsächlich alle Axiome aus CLIQUE^0 in $\mathbf{CLIQUE}^0$ erfüllt sind, d.h. $\mathbf{CLIQUE}^0 \in \mathit{Mod}(\text{CLIQUE}^0)$. Zum Nachweis der Monomorphie der Spezifikation verwenden wir Faktum 2.46 und müssen also zeigen, daß jede Grundgleichung durch CLIQUE^0 festgelegt wird. Wir unterscheiden hierzu die Grundgleichungen nach den Sorten der darin vorkommenden Terme:

1. $t_1^N = t_2^N$ mit Grundtermen t_1^N, t_2^N der Sorte `Nat`

   Zunächst zeigt man, daß für jeden Grundterm t^N der Sorte `Nat` ein Grundterm t_0^N der Form $\mathtt{succ}^n(\mathtt{zero})$ existiert mit

   $$\text{CLIQUE}^0 \models t^N = t_0^N.$$

   Hierzu verwendet man Induktion nach der Anzahl der Vorkommen von `first` oder `second` in t^N. Im Induktionsanfang kommt weder `first` noch `second` in t^N vor, und t^N ist schon von der gewünschten Form.

   Im Induktionsschritt betrachten wir das äußerste Auftreten des Funktionssymbols `first` oder `second` (wir nehmen im folgenden o.B.d.A. an, es handele sich um `first`; falls es sich um `second` handelt, läuft der Beweis völlig analog). Dann ist t^N von der Form $\mathtt{succ}^n(\mathtt{first}(t^P))$, wobei t^P Grundterm der Sorte `Nat-Pair` ist. Also ist t^P von der Form $\mathtt{pair}(t_F^N, t_S^N)$, und mit Axiom (1) (bzw. (2)) folgt

   $$\text{CLIQUE}^0 \models t^N = \mathtt{succ}^k(t_F^N).$$

   Für $\mathtt{succ}^k(t_F^N)$ wiederum ist die Induktionsvoraussetzung anwendbar und liefert die Existenz eines Terms t_0^N mit den gewünschten Eigenschaften.

   Wir können uns also im folgenden auf Grundgleichungen der Form $\mathtt{succ}^n(\mathtt{zero}) = \mathtt{succ}^m(\mathtt{zero})$ mit $m, n \in \mathbb{N}$ beschränken — alle Grundgleichungen der Sorte `Nat` lassen sich darauf zurückführen. Da NAT eine monomorphe Spezifikation ist, gilt für $m = n$, daß

   $$\text{CLIQUE} \models \mathtt{succ}^n(\mathtt{zero}) = \mathtt{succ}^m(\mathtt{zero}),$$

   und andernfalls

   $$\text{CLIQUE} \models \mathtt{succ}^n(\mathtt{zero}) \neq \mathtt{succ}^m(\mathtt{zero}),$$

   so daß für alle Grundgleichungen der Sorte `Nat` die Gültigkeit (bzw. Nicht-Gültigkeit) aus CLIQUE^0 folgerbar ist.

2. $t_1^G = t_2^G$ mit Grundtermen t_1^G, t_2^G der Sorte `Graph`

   Da `Graph` termerzeugt durch die Funktion `connect` ist, ist t_1^G von der Form $\texttt{connect}^{n_1}(\texttt{empty})$ und t_2^G von der Form $\texttt{connect}^{n_2}(\texttt{empty})$, wobei wir mit dieser Schreibweise kurz bezeichnen wollen, daß n_1 (bzw. n_2) Vorkommen der Funktion `connect` ineinander geschachtelt seien. Die Parameter der jeweiligen Anwendung von `connect` sind je zwei Terme $t_{1,n,1}^N, t_{1,n,2}^N$ für t_1^G bzw. $t_{2,n,1}^N, t_{2,n,2}^N$ für t_1^G.

   Nach der Argumentation für Grundgleichungen der Sorte `Nat` können wir uns im folgenden auf den Fall beschränken, daß alle $t_{i,n,j}^N$ von der Form $\texttt{succ}^m(\texttt{zero})$ sind. Ist

   $$\bigcup_{n=1}^{n_1} \left\{(t_{1,n,1}^N, t_{1,n,2}^N)\right\} \quad = \quad \bigcup_{n=1}^{n_2} \left\{(t_{2,n,1}^N, t_{2,n,2}^N)\right\},$$

   so kann man vermöge der Axiome (4) und (5) zeigen, daß

   $$\text{CLIQUE}^0 \quad \models \quad t_1^G = t_2^G.$$

   Andernfalls sei o.B.d.A (der andere Fall ist analog)

   $$(t_{1,j,1}^N, t_{1,j,2}^N) \quad \notin \quad \bigcup_{n=1}^{n_2}(t_{2,n,1}^N, t_{2,n,2}^N).$$

   Mit Hilfe der Axiome (3) und (4) kann man dann zeigen, daß

   $$\begin{array}{lll} \text{CLIQUE}^0 & \models & \neg hasedge(t_{1,j,1}^N, t_{1,j,2}^N, t_2^G), \text{ und} \\ \text{CLIQUE}^0 & \models & hasedge(t_{1,j,1}^N, t_{1,j,2}^N, t_1^G), \end{array}$$

   und die Kongruenzeigenschaft der Gleichheit liefert dann unmittelbar

   $$\text{CLIQUE}^0 \quad \models \quad t_1^G \neq t_2^G.$$

3. $t_1^P = t_2^P$ mit Grundtermen t_1^P, t_2^P der Sorte `Nat-Pair`

   t_1^P, t_2^P sind von der Form $\texttt{pair}(t_{1,1}^N, t_{1,2}^N)$ bzw. $\texttt{pair}(t_{2,1}^N, t_{2,2}^N)$. Sind aus CLIQUE^0 die Gleichungen $t_{1,1}^N = t_{2,1}^N$ und $t_{1,2}^N = t_{2,2}^N$ folgerbar, so erhält man aus der Kongruenz der Gleichheit sofort die Folgerbarkeit von $t_1^P = t_2^P$. Andernfalls sei o.B.d.A. $t_{1,1}^N = t_{2,1}^N$ nicht folgerbar. Nach Punkt (1) dieses Beweises ist aus CLIQUE^0 dann $t_{1,1}^N \neq t_{2,1}^N$ folgerbar und mit Hilfe von Axiom (1) (bzw. Axiom (2)) und der Kongruenz der Gleichheit auch $t_1^P \neq t_2^P$.

4. $t_1^B = t_2^B$ mit Grundtermen t_1^B, t_2^B der Sorte `Bool`

Entsprechend dem Vorgehen für Terme der Sorte `Nat` wollen wir auch hier zeigen, daß wir jeden Term t^B der Sorte `Bool` entweder auf `true` oder auf `false` „reduzieren" können, d.h. es gilt entweder

$$\begin{array}{rrcl} & \text{CLIQUE}^0 & \models & t^B = \texttt{true} \\ \text{oder} & \text{CLIQUE}^0 & \models & t^B = \texttt{false}. \end{array}$$

Ist t^B von der Form $\texttt{in}(t^N, t^G)$, so ist die Behauptung sofort aus der induktiven Axiomatisierung von `in` durch die Axiome (6) und (7) klar.

Ist t^B hingegen von der Form $\texttt{clique}(t^G)$, so betrachten wir Axiome (8) und (9). Unter Berücksichtigung der Tatsache, daß freie Variablen im Antezedenten der Implikation implizit existenzquantifiziert sind, erkennt man, daß der Antezedent in Axiom (8) das Negat des Antezedenten in Axiom (9) ist. Für alle in den Antzedenten vorkommenden Grundgleichungen haben wir schon nachgewiesen, daß entweder sie oder ihr Negat aus CLIQUE^0 folgerbar sind. Also läßt sich auch die Gültigkeit eines der beiden Antezedenten aus CLIQUE^0 folgern, und demnach entweder

$$\begin{array}{lcl} \text{CLIQUE}^0 & \models & \texttt{clique}(t^G) = \texttt{true} \\ \text{oder CLIQUE}^0 & \models & \texttt{clique}(t^G) = \texttt{false}. \end{array}$$

Wir können also jeden Term t^B der Sorte `Bool` entweder auf `true` oder `false` „reduzieren", und Grundgleichungen zwischen diesen beiden Konstanten sind durch CLIQUE^0 festgelegt, da sie in BOOL festgelegt sind.

Durch die Spezifikation CLIQUE werden also alle Grundgleichungen festgelegt, die Spezifikation ist demnach monomorph. ■

5.1.2 CLIQUE^0 $\notin$ Quan$_0$

In diesem Abschnitt wollen wir zeigen, daß keine monomorphe Spezifikation des Datentyps **CLIQUE**^{0} ohne Quantoren möglich ist. Sowohl die Aussage als auch deren Beweis ähneln sehr dem Lemma 4.5 — in unserem Fall lautet die Behauptung:

Lemma 5.6 (Nicht-*QF*-Spezifizierbarkeit von CLIQUE^0)
Es gibt keine QF-Spezifikation SP mit Mod(SP) $= [\mathbf{CLIQUE}^0]$.

Wir wählen wiederum unser Nicht-Standard-Modell $\mathbf{CLIQUE}^0_m$ so, daß es überall mit $\mathbf{CLIQUE}^0$ übereinstimmt außer

$$\texttt{clique}^{\mathbf{CLIQUE}^0_m}(G_m) \;=\; \mathit{false}.$$

Dann lautet unser entscheidendes Hilfslemma:

Lemma 5.7 (Nicht-Standard-Modelle)
Ist SP eine QF-Spezifikation mit $\mathbf{CLIQUE}^0 \in Mod(SP)$*, so existiert ein* $m \in \mathbb{N}$ *mit* $\mathbf{CLIQUE}^0_m \in Mod(SP)$.

Beweis von Lemma 5.6. Der Beweis von Lemma 5.6 verläuft hiermit völlig analog zu dem von Lemma 4.5, wir wollen daher auf eine Wiederholung verzichten. ■

Auch der Beweis von Lemma 5.7 stimmt zum großen Teil mit dem von Lemma 4.6 überein — es müssen allerdings wegen der reichhaltigeren Signatur mehr Fälle betrachtet werden. Um unnötige Wiederholungen zu vermeiden, wollen wir nur angeben, welche Veränderungen gegenüber dem Beweis von Lemma 4.6 auftreten. Bei allen Definitionen und Lemmata sind natürlich **CLIQUE** bzw. Σ_{CLIQUE} durch $\mathbf{CLIQUE}^0$ bzw. $\Sigma_{\mathrm{CLIQUE}^0}$ zu ersetzen.

Definitionen 4.7, 4.10, 4.14, 4.16, 4.17 und 4.20 wollen wir übernehmen, wobei bei der induktiven Definition in 4.10 und 4.14 für die nun größere Signatur noch folgende Fälle ergänzt werden müssen:

$$\begin{array}{lcl}
\mathrm{connects}(\mathtt{pair}(t_1^N, t_2^N)) & := & 0 \\
\mathrm{connects}(\mathtt{first}(t^P)) & := & 0 \\
\mathrm{connects}(\mathtt{second}(t^P)) & := & 0 \\
\mathrm{connects}(\mathtt{in}(t^N, t^G)) & := & \mathrm{connects}(t^G) \\
 & & \\
\max_v(\mathtt{pair}(t_1^N, t_2^N)) & := & \max(\max_v(t_1^N), \max_v(t_2^N)) \\
\max_v(\mathtt{first}(\mathtt{pair}(t_1^N, t_2^N))) & := & \max_v(t_1^N) \\
\max_v(\mathtt{second}(\mathtt{pair}(t_1^N, t_2^N))) & := & \max_v(t_2^N) \\
\max_v(\mathtt{in}(t^N, t^G)) & := & \max(\max_v(t^N), \max_v(t^G)).
\end{array}$$

Mit diesen Definitionen gelten dann die Lemmata 4.11, 4.12, 4.13, 4.15 und 4.18 genau wie für **CLIQUE**.

Hingegen sind bei den Lemmata 4.19 und 4.21 zusätzliche Fälle zu berücksichtigen. Wir wollen selbige daher nochmals formulieren und die Erweiterungen bei den Beweisen angeben.

Lemma 5.8 (cliquenfreie Literale in $\mathbf{CLIQUE}^0_m$)
Sei $v : X \to \mathbf{CLIQUE}^0$ *Variablenbelegung,* $m \in \mathbb{N}$ *und* λ *Literal, in dem kein Term* $\mathtt{clique}(t^G)$ *vorkommt, so daß* $(t^G)^{v,\mathbf{CLIQUE}^0} = G_m$. *Dann gilt*

$$\mathbf{CLIQUE}^0, v \models \lambda \implies \mathbf{CLIQUE}^0_m, v \models \lambda.$$

Beweis. Es werden die möglichen Formen des Literals λ unterschieden. Der Beweis für die Fälle, in denen λ eine Form aufweist, die auch unter der Signatur Σ_{CLIQUE} vorkommen kann, sind völlig identisch zu dem von Lemma 4.19. Wir beachten daher nur die zwei veränderten Fälle.

- $\underline{\lambda \equiv (t_1^P = t_2^P) \text{ oder } \lambda \equiv (t_1^P \neq t_2^P)}$:

  Aus der Definition von $\mathbf{CLIQUE}_m^0$ ist sofort ersichtlich, daß für alle Terme t^P der Sorte `Nat-Pair` gilt, daß

  $$(t^P)^{v,\mathbf{CLIQUE}^0} = (t^P)^{v,\mathbf{CLIQUE}_m^0},$$

  woraus die Behauptung unmittelbar folgt.

- $\underline{\lambda \equiv (t_1^B = t_2^B) \text{ oder } \lambda \equiv (t_1^B \neq t_2^B)}$:

  Auch hier wollen wir die Aussage zeigen, indem wir für jeden Term t^B der Sorte `Bool` nachweisen, daß

  $$(t^B)^{v,\mathbf{CLIQUE}^0} = (t^B)^{v,\mathbf{CLIQUE}_m^0}.$$

  Die Fälle, daß t^B von der Form `true`, `false` oder $\texttt{clique}(t^G)$ ist, sind schon im Beweis von Lemma 4.19 behandelt. Es bleibt der Fall, daß $t^B \equiv (\texttt{in}(t^N, t^G))$. Hierzu verwenden wir die ebenfalls in Lemma 4.19 bewiesenen Aussagen

  $$\begin{aligned}(t^N)^{v,\mathbf{CLIQUE}^0} &= (t^N)^{v,\mathbf{CLIQUE}_m^0} \text{ und} \\ (t^G)^{v,\mathbf{CLIQUE}^0} &= (t^G)^{v,\mathbf{CLIQUE}_m^0}.\end{aligned}$$

  Mit der Eigenschaft

  $$\texttt{in}^{\mathbf{CLIQUE}^0}(x, G) = \texttt{in}^{\mathbf{CLIQUE}_m^0}(x, G)$$

  aus der Definition von $\mathbf{CLIQUE}_m^0$ folgt dann die behauptete Gleichheit der Auswertung für jeden einzelnen Term der Sorte `Bool`. Hieraus erhalten wir wiederum, daß auch jedes Literal der Sorte `Bool` in beiden Algebren gleich ausgewertet wird.

∎

Lemma 5.9 (cliquenfreie Literale)
Sei $\varphi = \bigvee_{j=1}^J \lambda_j$, $v : X \to \mathbf{CLIQUE}^0$ *eine Variablenbelegung, weiterhin* $m \geq \text{connects}(\varphi)$ *und* $n \geq \max_v(\varphi)$. *Es gelte für alle* $j \in \{1, \ldots, J\}$: *Ist* λ_j *kein* $[U_{\varphi,v,m}]_{\sim_\varphi}$*-Cliquenliteral, so* $\mathbf{CLIQUE}^0, v \not\models \lambda_j$.

Dann gilt auch für alle $j \in \{1, \ldots, J\}$, *die keine* $[U_{\varphi,v,m}]_{\sim_\varphi}$*-Cliquenliterale sind, daß*

$$\mathbf{CLIQUE}^0, v_{\varphi,m,n} \not\models \lambda_j$$

Beweis. Auch hier genügt es, die Fälle zu betrachten, die gegenüber Lemma 4.21 hinzukommen:

- $\underline{\lambda_j \equiv (t_1^P = t_2^P) \text{ oder } \lambda_j \equiv (t_1^P \neq t_2^P)}$:

  Nach Lemma 4.18 gilt für jeden Term t^N der Sorte `Nat`, daß

  $$(t^N)^{v,\mathbf{CLIQUE}^0} \;=\; (t^N)^{v_{\varphi,m,n},\mathbf{CLIQUE}^0}.$$

  Mit der Definition von $\texttt{Nat-Pair}^{\mathbf{CLIQUE}^0}$ folgt dann sofort, daß auch für jeden Term t^P der Sorte `Nat-Pair` gilt:

  $$(t^P)^{v,\mathbf{CLIQUE}^0} \;=\; (t^P)^{v_{\varphi,m,n},\mathbf{CLIQUE}^0},$$

  was schließlich die Behauptung impliziert.

- $\underline{\lambda_j \equiv (t_1^B = t_2^B) \text{ oder } \lambda_j \equiv (t_1^B \neq t_2^B)}$:

  Wir wollen zeigen, daß für jeden Term t^B der Sorte `Bool` gilt, daß

  $$(t^B)^{v,\mathbf{CLIQUE}^0} \;=\; (t^B)^{v_{\varphi,m,n},\mathbf{CLIQUE}^0}.$$

  Hierzu müssen wir nur noch den gegenüber 4.21 hinzugekommenen Fall betrachten, daß t^B von der Form $\texttt{in}(t^N, t^G)$ ist.

  Nach Definition von $\max_v$ und Voraussetzung ist dann

  $$(t^N)^{v,\mathbf{CLIQUE}^0} \;\leq\; n \;<\; n+1,$$

  d.h. insbesondere ist $(t^N)^{v,\mathbf{CLIQUE}^0} \notin \{n+1, n+2\}$. Damit gilt dann

  $$\begin{aligned} &(t^N)^{v,\mathbf{CLIQUE}^0} \in (t^G)^{v,\mathbf{CLIQUE}^0} \\ &\quad\Longleftrightarrow\quad (t^N)^{v_{\varphi,m,n},\mathbf{CLIQUE}^0} \in (t^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}^0}, \end{aligned}$$

  also folgt die Behauptung.

■

Lemmata 4.22 und 4.23 können mitsamt Beweisen wörtlich übernommen werden, wobei natürlich anstelle der Lemmata 4.19 und 4.21 die Lemmata 5.8 und 5.9 verwendet werden müssen.

Damit können wir schließlich den Beweis zu Lemma 5.7 angeben, der auch direkt an den Beweis von Lemma 4.6 angelehnt ist.

Beweis von Lemma 5.7. Sei SP eine beliebige QF-Spezifikation über der Signatur Σ_{CLIQUE^0}. Wir bilden die Konjunktion aller Axiome und bringen die resultierende Formel auf konjunktive Normalform. Das Ergebnis sei mit ϕ bezeichnet. Mit $m \geq \text{connects}(\phi)$ sind dann die Voraussetzungen von Lemma 4.23 (bzw. dessen Modifikation) erfüllt, und man erhält

$$\begin{aligned}\mathbf{CLIQUE}^0 \in Mod(SP) &\implies \mathbf{CLIQUE}^0 \models \phi \\ &\implies \mathbf{CLIQUE}^0_m \models \phi \\ &\implies \mathbf{CLIQUE}^0_m \in Mod(SP).\end{aligned}$$

■

5.2 Spezifikation mit mindestens zwei Quantoren

In diesem Abschnitt werden wir das Beispiel $\mathbf{CLIQUE}^0$ so modifizieren, daß die resultierenden Datentypen $\mathbf{CLIQUE}^k$ jeweils mit $k+1$, nicht jedoch mit k Quantoren monomorph spezifizierbar sind.

Die Grundidee dabei ist, den Datentypen $\mathbf{CLIQUE}^0$ quasi zu duplizieren. Wir vervielfältigen die Signatur Σ_{CLIQUE^0} insgesamt $k+1$ mal und wählen unseren Datentypen so, daß er auf den jeweiligen Teilsignaturen mit $\mathbf{CLIQUE}^0$ übereinstimmt. Da wir keine Funktionen zwischen den Sorten verschiedener Duplikate bereitstellen, können wir erreichen, daß der Datentyp $\mathbf{CLIQUE}^0$ gleichsam $k+1$ mal spezifiziert werden muß.

Formell definieren wir zu jedem $n \in \mathbb{N}$ die Signatur

signature Σ_n

sorts	$\texttt{Bool}_n, \texttt{Nat}_n, \texttt{Nat-Pair}_n, \texttt{Graph}_n$		
functions	$\texttt{true}_n, \texttt{false}_n$	:	$\rightarrow \texttt{Bool}_n$
	$\texttt{zero}_n$	:	$\rightarrow \texttt{Nat}_n$
	$\texttt{succ}_n$	: $\texttt{Nat}_n$	$\rightarrow \texttt{Nat}_n$
	$\texttt{pair}_n$	: $\texttt{Nat}_n \times \texttt{Nat}_n$	$\rightarrow \texttt{Nat-Pair}_n$
	$\texttt{first}_n$	: $\texttt{Nat-Pair}_n$	$\rightarrow \texttt{Nat}_n$
	$\texttt{second}_n$	: $\texttt{Nat-Pair}_n$	$\rightarrow \texttt{Nat}_n$
	$\texttt{empty}_n$	:	$\rightarrow \texttt{Graph}_n$
	$\texttt{connect}_n$	: $\texttt{Nat}_n \times \texttt{Nat}_n \times \texttt{Graph}_n$	$\rightarrow \texttt{Graph}_n$
	$\texttt{in}_n$	: $\texttt{Nat}_n \times \texttt{Graph}_n$	$\rightarrow \texttt{Bool}_n$
	$\texttt{clique}_n$	: $\texttt{Graph}_n$	$\rightarrow \texttt{Bool}_n$

end signature

und damit die Signatur

$$\begin{aligned}\Sigma_{\mathrm{CLIQUE}^{\mathrm{k}}} &= \bigcup_{n=0}^{k} \Sigma_n \\ &= \left(\bigcup_{n=0}^{k} S_k, \bigcup_{n=0}^{k} F_k\right).\end{aligned}$$

Um die nachfolgenden Argumenationen formal korrekt durchführen zu können, definieren wir für jedes n einen Signaturmorphismus (vgl. Abschnitt 2.1.5) $\sigma_n : \Sigma_{\mathrm{CLIQUE}^0} \to \Sigma_n$ den, der jedes Sortensymbol s auf s_n und jedes Funktionssymbol f auf f_n abbildet (σ_n ist offenbar sogar bijektiv).

Damit können wir nun die gewünschte $\Sigma_{\mathrm{CLIQUE}^{\mathrm{k}}}$-Algebra $\mathbf{CLIQUE}^k$ definieren durch

$$\mathbf{CLIQUE}^k|_{\sigma_n} = \mathbf{CLIQUE}^0 \text{ für jedes } n \leq k.$$

Dann können wir die zentrale Aussage dieses Abschnitts folgendermaßen formulieren:

Lemma 5.10 (Spezifizierbarkeit von $\mathbf{CLIQUE}^k$)
Für jedes $k \geq 1$ ist $\mathbf{CLIQUE}^k \in \mathrm{Quan}_{k+1} \setminus \mathrm{Quan}_k$.

Auch hier werden wir die beiden Teilaussagen $\mathbf{CLIQUE}^k \in \mathrm{Quan}_{k+1}$ und $\mathbf{CLIQUE}^k \notin \mathrm{Quan}_k$ in getrennten Abschnitten nachweisen.

5.2.1 $\mathbf{CLIQUE}^k \in \mathbf{Quan}_{k+1}$

Um zu zeigen, daß $\mathbf{CLIQUE}^k \in \mathrm{Quan}_{k+1}$, geben wir wiederum eine geeignete Spezifikation CLIQUE^k an und weisen nach, daß diese den Datentypen monomorph spezifiziert.

Es sei CLIQUE_n die Spezifikation, die aus CLIQUE^0 entsteht, indem jedes Auftreten eines Sortensymbols s aus $\Sigma_{\mathrm{CLIQUE}^0}$ durch $\sigma_n(s)$ und entsprechend jedes Auftreten eines Funktionssymbols f aus $\Sigma_{\mathrm{CLIQUE}^0}$ durch $\sigma_n(f)$ ersetzt wird.

Dann setzen wir

$$\mathrm{CLIQUE}^k = \bigcup_{n=0}^{k} \mathrm{CLIQUE}_n,$$

wobei die Vereinigung wiederum komponentenweise zu verstehen ist.

Da jedes der CLIQUE_n genau einen Quantor enthält, enthält CLIQUE^k genau $k+1$ Quantoren. Wir müssen also zum Nachweis unserer Behauptung noch die folgende Aussage zeigen:

Lemma 5.11 ($\mathbf{CLIQUE}^k$ spezifiziert $\mathbf{CLIQUE}^k$)
$Mod(\mathrm{CLIQUE}^k) = [\mathbf{CLIQUE}^k]$.

Beweis. Daß $\mathbf{CLIQUE}^k$ tatsächlich Modell von CLIQUE^k ist, folgt daraus, daß $\mathbf{CLIQUE}^0$ Modell von CLIQUE^0 ist. Wir müssen also nach Faktum 2.46 noch zeigen, daß für jede Grundgleichung sie oder ihr Negat aus CLIQUE^k folgt.

Hierzu unterscheiden wir wiederum nach den verschiedenen Sorten der Grundgleichungen. Für Grundgleichungen λ_n der Sorte s_n (d.h. $\mathtt{Bool}_n$, $\mathtt{Nat}_n$, $\mathtt{Nat\text{-}Pair}_n$ oder $\mathtt{Graph}_n$) zeigt man mit Hilfe von Lemma 5.5 und unter Anwendung von σ_n, daß

$$\begin{array}{rcl} \mathrm{CLIQUE}_n & \models & \lambda_n \\ \text{bzw. } \mathrm{CLIQUE}_n & \models & \neg\lambda_n. \end{array}$$

Damit erhält man dann insbesondere

$$\begin{array}{rcl} \mathrm{CLIQUE}^k & \models & \lambda_n \\ \text{bzw. } \mathrm{CLIQUE}^k & \models & \neg\lambda_n. \end{array}$$

Also folgt für jede Grundgleichung sie oder ihr Negat aus CLIQUE^k, und CLIQUE^k ist demnach monomorph. ■

5.2.2 $\mathbf{CLIQUE}^k \notin \mathbf{Quan}_k$

In diesem Abschnitt wollen wir nachweisen, daß für jedes $k \geq 1$ der Datentyp $\mathbf{CLIQUE}^k$ nicht mit k oder weniger Quantoren monomorph spezifiziert werden kann. Wir werden also folgendes Lemma beweisen:

Lemma 5.12 (Nicht-Spezifizierbarkeit von $\mathbf{CLIQUE}^k$)
Sei $k \geq 1$. Dann gibt es keine PL-Spezifikation SP mit k oder weniger Quantoren, so daß

$$Mod(SP) \;=\; [\mathbf{CLIQUE}^k].$$

Zum Beweis des Lemmas konstruieren wir wiederum Nicht-Standard-Modelle der gleichen Form wie auch schon in Kapitel 4, und zwar unter Modifikation der Funktion $\mathtt{clique}_n$ für genau ein n. Wir definieren für jedes $m \in \mathbb{N}$ und $n \leq k$ einen Datentyp $\mathbf{CLIQUE}^k_{m,n}$ so, daß alle Sorten und Funktionen mit denen in $\mathbf{CLIQUE}^k$ übereinstimmen außer

$$\mathtt{clique}_n^{\mathbf{CLIQUE}^k_{m,n}}(G) \;=\; \begin{cases} \mathit{false} & \text{, für } G = G_m \\ \mathtt{clique}_n^{\mathbf{CLIQUE}^k}(G) & \text{, sonst} \end{cases}$$

Damit können wir unser Lemma über die Existenz von Nicht-Standard-Modellen formulieren:

Lemma 5.13 (Nicht-Standard-Modelle)
Sei $k \geq 1$ und SP eine PL-Spezifikation mit höchstens k Quantoren. Gilt dann $\mathbf{CLIQUE}^k \in Mod(SP)$, so existieren $m \in \mathbb{N}$ und $n \leq k$ mit

$$\mathbf{CLIQUE}^k_{m,n} \quad \in \quad Mod(SP).$$

Beweis von Lemma 5.12. Mit Hilfe von Lemma 5.13 ist der Beweis von Lemma 5.12 einfach. Sei SP eine PL-Spezifikation mit höchstens k Quantoren. Ist $\mathbf{CLIQUE}^k \in Mod(SP)$, so folgt mit Lemma 5.13, daß für ein $m \in \mathbb{N}$ und ein $n \leq k$ auch $\mathbf{CLIQUE}^k_{m,n} \in Mod(SP)$. Insbesondere erhalten wir dann $Mod(SP) \neq [\mathbf{CLIQUE}^k]$. Ist hingegen $\mathbf{CLIQUE}^k \notin Mod(SP)$, so folgt trivialerweise $Mod(SP) \neq [\mathbf{CLIQUE}^k]$. ∎

Beweis von Lemma 5.13. Den Beweis von Lemma 5.13 wollen wir nicht in allen Details durchführen, da er nur technischen Aufwand, nicht jedoch neue Beweisideen mit sich bringen würde.

Sei SP eine Spezifikation mit höchstens k Quantoren. Durch Umbenennung der Variablen können wir die Axiome paarweise variablendisjunkt wählen, anschließend die Konjunktion aller Axiome bilden und die resultierende Formel in Pränex-Normalform bringen. Wir erhalten eine zu den ursprünglichen Axiomen äquivalente Formel ϕ.

Da ϕ höchstens k Quantoren enthält, muß mindestens ein $n \in \{0, \ldots, k\}$ existieren, so daß keine Variable der Sorten $\mathtt{Bool}_n, \mathtt{Nat}_n, \mathtt{Nat\text{-}Pair}_n, \mathtt{Graph}_n$ durch einen Quantor gebunden ist (es müssen in SP nicht notwendig Variablen der entsprechenden Sorten auftreten). Wir fixieren dieses n im folgenden und wählen ein $m > \mathrm{connects}_n(\phi)$, wobei die Anzahl von $\mathtt{connect}_n$ entsprechend der Anzahl von `connect` definiert sei (vgl. Definition 4.10).

Für das entsprechende Nicht-Standard-Modell $\mathbf{CLIQUE}^k_{m,n}$ führen wir nun eine völlig analoge Argumentation durch wie für $\mathbf{CLIQUE}^0_{m,n}$ in Abschnitt 5.1. Dies ist möglich, da alle Terme der Sorten s_j mit $j \neq n$ in $\mathbf{CLIQUE}^k_{m,n}$ und $\mathbf{CLIQUE}^k$ gleich ausgewertet werden. Die Modifikation der Variablenbelegung wird auch nur für Variablen der Sorte $\mathtt{Graph}_n$ vorgenommen, und zwar lediglich in Abhängigkeit von Variablen der Sorten s_n, so daß nicht indirekt Werte quantifizierter Variablen in die Modifikation der freien Variablen eingehen.

Auf diese Weise zeigt man, daß für alle Variablenbelegungen v auch

$$\mathbf{CLIQUE}^k_{m,n}, v \quad \models \quad \phi,$$

und eine Argumentation entsprechend dem Beweis von Lemma 4.6 auf Seite 52 zeigt die Behauptung. ∎

5.3 Zusätzliche Randbedingungen

In den letzten beiden Abschnitten haben wir einen Beweis von Satz 5.3 geführt und damit gezeigt, daß es tatsächlich Datentypen gibt, zu deren monomorpher Spezifikation beliebig viele Quantoren benötigt werden. Allerdings ist die dabei verwendete Beweistechnik in gewisser Weise unbefriedigend, da die benötigte Anzahl Quantoren weniger von der *„Schwierigkeit“* als von der *„Größe“* des Datentyps herrührt.

Man würde sich jedoch wünschen, die Anzahl von zur monomorphen Spezifikation benötigten Quantoren als Maß für die „Schwierigkeit“ eines Datentyps ansehen zu können. Wir wollen an dieser Stelle keinen Versuch machen, den informellen Begriff „Schwierigkeit“ mathematisch zu präzisieren — in Abschnitt 6.3 werden wir jedoch zeigen, daß die naheliegende Idee, die Komplexität (als algebraische oder Berechnungskomplexität) zur Präzisierung zu verwenden, zumindest nicht unmittelbar geeignet ist, einen Zusammenhang zur Anzahl benötigter Quantoren zu erhalten.

5.3.1 Beschränkte Zahl von Sorten

Um Konstruktionen wie die in Abschnitt 5.2 auszuschließen, wäre eine naheliegende Idee die, die Anzahl der Sorten der Datentypen durch eine Konstante zu beschränken. Eine Modifikation von Satz 5.3 sähe dann wie folgt aus:

Vermutung 5.14 (Hierarchie mit beschränkter Anzahl Sorten)
Es gibt ein $m_s \in \mathbb{N}$, so daß für jedes $k \in \mathbb{N}$ ein Datentyp A_k mit höchstens m_s Sorten existiert, so daß:

$$A_k \quad \in \quad \mathrm{Quan}_{k+1} \setminus \mathrm{Quan}_k.$$

Zum Beweis können wir dann, wie gewünscht, die Konstruktion aus Abschnitt 5.2 nicht mehr anwenden. Allerdings wird der Beweis des Satzes auch deutlich schwerer als der von Satz 5.3, und im Rahmen dieser Arbeit ist kein vollständiger Beweis gelungen. Dennoch wollen wir im folgenden eine aussichtsreiche Beweisidee und die dabei aufgetretenen Schwierigkeiten skizzieren.

Eine Möglichkeit, den Datentypen **CLIQUE** so zu modifizieren, daß zur Spezifikation mehr Quantoren benötigt werden, wäre eine Verallgemeinerung von Graphen auf Hypergraphen, d.h. die Kanten verlaufen nicht zwischen je zwei, sondern zwischen drei, vier oder allgemein k Knoten. Wir sehen dann — analog zur Definition 4.1 — Hypergraphen als endliche Teilmengen von $\mathbb{N}^k$ an, und bezeichnen einen Hypergraphen G als vollständig, wenn für jedes

k-Tupel, dessen Knoten in G enthalten sind, auch das k-Tupel der Knoten in G enthalten ist.

Zum Beweis der Aussage für ein $k > 0$ können wir dann den entsprechenden Datentyp für $(k+1)$-Graphen wählen (wobei wir keine Sorten und Funktionen für Paare natürlicher Zahlen vorsehen wollen). Eine Spezifikation mit $k+1$ Quantoren läßt sich leicht durch Modifikation der Spezifikation CLIQUE^0 erhalten. Um zu zeigen, daß der resultierende Datentyp nicht mit k Quantoren spezifiziert werden kann, müssen wieder Nicht-Standard-Modelle konstruiert werden.

Die Grundidee eines Beweises ist wieder die, daß die Vollständigkeit eine globale Eigenschaft eines Hypergraphen ist, d.h. es müssen zur Sicherung der Vollständigkeit Aussagen über alle $(k+1)$-Tupel von Knoten gemacht werden, wofür man (vermutlich) $k+1$ Quantoren benötigt. Mit k Quantoren können insbesondere höchstens k Variablen der Sorte `Nat` gebunden werden, so daß man — anschaulich gesprochen — nur Aussagen über k-dimensionale Unterräume des $\mathbb{N}^{k+1}$ machen kann.

Anhand syntaktischer Kriterien der Spezifikation SP (unter anderem die Anzahl der Vorkommen von `connect`) wählt man einen „großen“ vollständigen Hypergraphen G_m, der im Nicht-Standard-Modell durch die Funktion `clique` als nicht vollständig charakterisiert wird. Man muß dann zeigen, daß jedes Axiom der Spezifikation im Nicht-Standard-Modell ebenfalls zu wahr ausgewertet wird.

Um für jedes Axiom seine Allgemeingültigkeit im Nicht-Standard-Modell zu zeigen, kann man die Axiome in Pränex-Normalform (vgl. Definition 2.37) betrachten und Skolem-Funktionen im Nicht-Standard-Modell anhand der Skolem-Funktionen des Standard-Modells definieren. Anschließend muß gezeigt werden, daß alle im Standardmodell unter der entsprechenden Variablenbelegung zu wahr ausgewerteten Literale im Nicht-Standard-Modell ebenfalls zu wahr ausgewertet werden.

Während ein solcher Ansatz zunächst sehr plausibel erscheint, ergeben sich Schwierigkeiten dadurch, daß auch Quantoren über Variablen der Sorte `Graph` (die wir hier für $(k+1)$-Graphen verwenden wollen) auftreten können. Denn um sicherzustellen, daß im Nicht-Standard-Modell mindestens die Literale zu wahr ausgewertet werden, die auch im Standardmodell zu wahr ausgewertet werden, müssen wir verhindern, daß bei einem Literal der Form $\texttt{clique}(t^G) = \texttt{true}$ der Term t^G zu G_m ausgewertet wird.

Die Konstruktion der Variablenmodifikation aus Definition 4.17 hat ein ähnliches Ziel gehabt. Durch die hinzugefügte Kante wurde sichergestellt, daß ein Literal der Form $\texttt{clique}(t^G) = \texttt{true}$ auch im Standardmodell unter der modifizierten Variablenbelegung zu *false* ausgewertet wurde. Dort stand allerdings Information (in Form der Variablenbelegung) über die Werte *aller*

Variablen zur Verfügung, während in unserem Fall die Belegung der Variablen so gewählt werden muß, daß die Gültigkeit für alle Belegungen der später auftretenden gebundenen Variablen gesichert ist.

Im Fall freier oder allquantifizierter Variablen können wir noch ähnlich vorgehen wie im Beweis in Abschnitt 4.3 und Kanten entfernen. Ein Hinzufügen von Kanten ist nicht empfehlenswert, da wir ja keine geeignete Größe wie $\max_v$ zur Verfügung haben. Werden die Knoten geschickt gewählt (vor allem so, daß keiner der Knoten der Wert eines Terms ist, der eine freie Variable der Sorte `Nat` enthält — auf weitere Details wollen wir hier nicht eingehen), kann ihr Fehlen von der Formel nicht „entdeckt“ werden.

Problematischer ist die Situation jedoch, wenn eine Variable der Sorte `Graph` durch einen Existenzquantor gebunden ist. Wenn wir hier eine entsprechende Modifikation vornehmen, ist nicht notwendigerweise gesichert, daß mit der neuen Belegung die Formel im Standard-Modell zu wahr ausgewertet wird. Die Aufgabe, eine entsprechende Modifikation für die Wahl des Wertes der Variablen im Nicht-Standard-Modell vorzunehmen und nachfolgenden Variablenbelegungen so anzupassen, daß die Gültigkeit garantiert ist, ist bis jetzt nicht gelöst.

Angesichts der Schwierigkeiten, die bei intensiver Beschäftigung mit dem Beweis auftreten, mag man auch vermuten, daß das obige Beispiel vielleicht gar kein Vertreter der entsprechenden Klasse von Datentypen ist. Allerdings ist auch nicht offensichtlich, wie eine entsprechende Spezifikation aussehen könnte. Das Problem, ob sich der Datentyp mit k Quantoren spezifizieren läßt, bleibt also hier ungelöst stehen, womit natürlich auch Vermutung 5.14 noch nicht bewiesen ist.

5.3.2 Beschränkte Stelligkeit von Funktionen

Ungeachtet der Tatsache, daß die Frage der Gültigkeit der Vermutung 5.14 noch nicht geklärt ist, mag man noch strengere Anforderungen an die Vertreter einer Hierarchie stellen. Denn selbst wenn ein Beweis wohl schwierig wäre und bis jetzt nicht geglückt ist, so waren doch auch die Datentypen, die im vorangegangenen Abschnitt beschrieben wurden, nicht wirklich „schwieriger“, sondern auch nur „größer“ — schließlich wächst die Stelligkeit der Funktion `connect` linear in k.

Um auch solche Konstruktionen zu vermeiden, liegt es nahe, zusätzlich zur Beschränktheit der Anzahl Sorten zu fordern, daß auch die Anzahl Funktionen und die Stelligkeit aller Funktionen der entsprechenden Datentypen durch Konstanten beschränkt bleiben. Man erhält dann folgende Version:

Vermutung 5.15 (Hierarchie für beschränkte Stelligkeit)
Es gibt $m_s, m_f, m_a \in \mathbb{N}$, *so daß für jedes* $k \in \mathbb{N}$ *eine Signatur* $\Sigma_k = (S_k, F_k)$ *und ein* Σ_k*-Datentyp* A_k *existieren mit:*

1. $A_k \in \mathrm{Quan}_{k+1} \setminus \mathrm{Quan}_k$.
2. $|S_k| \leq m_s$.
3. $|F_k| \leq m_f$.
4. *Jedes Funktionssymbol* $f \in F_k$ *hat höchstens Stelligkeit* m_a.

Obwohl schon der Beweis von Vermutung 5.14 in dieser Arbeit nicht geführt wird, scheint Vermutung 5.15 noch ungleich schwieriger, da er eben keine direkte Verallgemeinerung eines einmal gewählten Beispiels auf höherdimensionale, aber ansonsten identische, Probleme zuläßt. Während die Aussage von Vermutung 5.14 wohl dem entspricht, was man bezüglich dieser Problemstellung vermuten würde, ist die Gültigkeit von Vermutung 5.15 deutlich fragwürdiger. Die Stärke der Aussage wird noch offensichtlicher in Form von

Vermutung 5.16 (Hierarchie mit fester Signatur)
Es gibt eine Signatur Σ, *so daß für jedes* $k \in \mathbb{N}$ *ein* Σ*-Datentyp* A_k *existiert, so daß*

$$A_k \quad \in \quad \mathrm{Quan}_{k+1} \setminus \mathrm{Quan}_k.$$

Tatsächlich sind beide Aussagen äquivalent, wie wir nachfolgend kurz beweisen wollen: Die Implikation von 5.16 nach 5.15 ist trivial. Für die andere Richtung stellt man fest, daß es (bis auf Umbenennung der Signatursymbole) zu gegebenen m_s, m_f, m_a nur endlich viele Signaturen mit höchstens m_s Sortensymbolen und m_f Funktionssymbolen der Stelligkeit höchstens m_a gibt. Wählt man für alle diese Signaturen die Bezeichner für Sorten und Funktionen paarweise disjunkt und bildet die Vereinigung aller Signaturen, so erhält man die gewünschte Signatur Σ.

Zu jedem Datentyp A_k mit Signatur Σ_k aus 5.15 kann man nun einen Σ-Datentyp A'_k wählen mit $A'_k|_{\Sigma_k} = A_k$ und so, daß alle anderen Sorten durch die Funktionen entsprechender Zielsorte frei erzeugt sind. Für deren Spezifikation sind dann keine Quantoren nötig (vgl. Abschnitt 6.2), und mit einem Beweis entsprechend dem aus Abschnitt 5.2 kann man dann zeigen, daß der resultierende Datentyp die gewünschten Eigenschaften hat.

Aus der Formulierung als Vermutung 5.16 ergibt sich wiederum eine verwandte Fragestellung, nämlich die, ob zu einer *fest vorgegebenen* Signatur

Σ eine Hierarchie von Datentypen bezüglich der zur Spezifikation benötigten Anzahl von Quantoren existiert,[3] bzw. wie reichhaltig eine Signatur sein muß, damit die Existenz einer Hierarchie gesichert ist. Beispielsweise existiert keine solche Hierarchie für eine Signatur Σ_C, die nur Konstanten (also 0-stellige Funktionen) enthält, da jeder Σ_C-Datentyp endlich und somit quantorenfrei spezifizierbar wäre (vgl. Abschnitt 6.2).

5.3.3 Andere Hierarchien

Obwohl wir in den letzten Abschnitten verschiedene alternative Ansätze für Hierarchieergebnisse betrachtet haben, haben wir stets daran festgehalten, Eigenschaften der oben definierten Quan_k (ggf. mit zusätzlichen Randbedingungen wie Beschränkung der Sortenzahl, Fixierung der Signatur usw.) zu untersuchen.

Anstelle der *Anzahl* der in Spezifikationen vorkommenden Quantoren sind auch durchaus andere Maße denkbar. Insbesondere die maximale Schachtelungstiefe bietet sich hier an. Selbst ohne die Forderung nach beschränkter Anzahl Sorten ist dann nicht mehr die Konstruktion der „Vervielfachung" von Datentypen aus Abschnitt 5.2 möglich. Der dort konstruierte Datentyp ist mit k Allquantoren (einen für jedes „Duplikat") und Schachtelungstiefe 1 spezifizierbar, indem ein Axiom im Stile von Axiom (9) der Spezifikation CLIQUE^0 (auf Seite 56) hinzugefügt wird, bei dem der Antezedent die Konjunktion der entsprechend umbenannten Vorbedingungen ist.

Zuletzt wollen wir noch einen weiteren Ansatz erwähnen, da entsprechende Hierarchien in Bereichen wie der Rekursionstheorie (vgl. etwa [Odi89] oder [Rog67]) oder der Modelltheorie (vgl. [Rob74]) üblich sind. Hierzu gehen wir davon aus, daß die Spezifikation (bis auf Generiertheitsbedingungen) aus einem einzigen Axiom in Pränex-Normalform besteht,[4] d.h. das Axiom ist von der Form

$$\phi \;\equiv\; (Q_1 x_1.\,(\ldots(Q_n x_n.\,(\varphi))\ldots)),$$

wobei jedes Q_j entweder '$\forall$' oder '$\exists$' und φ quantorenfrei ist. Dann kann man als Maß die Anzahl der Alterationen zwischen '$\forall$' und '$\exists$' wählen, d.h. die Anzahl der $k \in \{1,\ldots,n-1\}$, so daß $Q_k \neq Q_{k+1}$.

[3] Diese Fragestellung ist nur interessant, wenn man die Gültigkeit von 5.16 voraussetzt bzw. beweist — andernfalls ist die Frage für jedes Σ mit „Nein" zu beantworten.

[4] Eine solche Form existiert für jede Spezifikation — man wähle die Namen der Variablen der einzelnen Axiome paarweise disjunkt und bilde die Konjunktion aller Axiome. Für die resultierende Formel existiert nach Faktum 2.39 eine äquivalente Formel in Pränex-Normalform.

Insbesondere in der Modelltheorie (vgl. etwa [Rob74]) werden Eigenschaften der Modelle von Formeln mit entsprechend vielen Alterationen untersucht. Dort werden die Modelle weder auf termerzeugte eingeschränkt noch wird die Monomorphie von Datentypen gefordert (vgl. Abschnitt 6.4), so daß die Ergebnisse nicht direkt übertragbar sind. Ebenfalls über die Anzahl der Quantoren-Alterationen wird im Bereich der Rekursionstheorie die arithmetische Hierarchie definiert, wobei dort der Bereich des Berechenbaren verlassen wird, so daß die Ergebnisse in unserem Zusammenhang weniger von Interesse sind.

Trotzdem wäre die Untersuchung der Eigenschaften entsprechender Hierarchien (ggf. wieder mit den obigen Rahmenbedingungen) im Bereich der monomorphen Spezifikation termerzeugter Datentypen an sich interessant. Insbesondere könnte man die Existenz von Zusammenhängen zwischen den einzelnen Hierarchien untersuchen.

Kapitel 6

Bedingungen für QF-Spezifizierbarkeit

Nachdem wir in Kapitel 4 nachgewiesen haben, daß unter loser Semantik mit voller Prädikatenlogik echt mehr Datentypen monomorph spezifiziert werden können als mit quantorenfreien Axiomen, wollen wir uns in diesem Kapitel der Frage zuwenden, inwieweit sich die quantorenfreie Spezifizierbarkeit durch andere Eigenschaften eines Datentyps charakterisieren läßt.

Wir sind in dieser Arbeit weit davon entfernt, eine vollständige Charakterisierung geben zu können. Tatsächlich beziehen sich unsere positiven Resultate — sowohl was quantorenfreie Spezifizierbarkeit als auch Nicht-QF-Spezifizierbarkeit betrifft — nur auf sehr eingeschränkte Klassen von Datentypen. Ein in Abschnitt 6.4 vorgestelltes algebraisches Kriterium ist bei der Spezifikation monomorpher Datentypen eher selten anwendbar.

Andererseits werden wir in den Abschnitt 6.3 und 6.5 vielversprechende Ansätze betrachten und zeigen, daß diese in der betrachteten Form *nicht* zur Charakterisierung geeignet sind. Inwieweit andere Eigenschaften der Datentypen (insbesondere algebraische Eigenschaften) hinreichende oder notwendige Bedingungen für QF-Spezifizierbarkeit liefern, mag Thema zukünftiger Forschungsarbeit sein.

6.1 Eigenschaften von Graphen

In diesem Abschnitt wollen wir allerdings zunächst eine etwas größere Klasse von Datentypen vorstellen, die alle nicht quantorenfrei spezifizierbar sind. Wir verallgemeinern dazu das Beispiel **CLIQUE** aus Kapitel 4 und geben ein hinreichendes Kriterium dafür an, daß eine Eigenschaft von Graphen nicht quantorenfrei spezifizierbar ist. Dazu wollen wir zunächst die entscheidende Eigenschaft einer Menge von Graphen definieren:

Definition 6.1 (einsame Graphen)
Sei $\mathcal{M} \subseteq \mathcal{G}$ eine Menge von Graphen. Für $n \in \mathbb{N}$ nennen wir einen Graphen G **n-einsam in $\mathcal{M}$**, falls für jedes $G' \in \mathcal{M}$ gilt:

$$G \neq G' \implies d(G, G') \geq n.$$

Wir sagen, die Menge $\mathcal{M}$ **enthält beliebig einsame Graphen**, falls zu jedem $n \in \mathbb{N}$ ein n-einsamer Graph in $\mathcal{M}$ existiert und $|\mathcal{M}| > 1$.[1] □

Aus der Definition ist offensichtlich, daß ein in $\mathcal{M}$ n-einsamer Graph G auch m-einsam in $\mathcal{M}$ ist für jedes $m \leq n$.

Beispiel 6.2 (Mengen mit einsamen Graphen)
Um den Begriff der n-Einsamkeit etwas zu illustrieren, wollen wir hier einige Beispiele für Mengen angeben, die beliebig einsame Graphen enthalten, sowie für Mengen, die keine beliebig einsamen Graphen enthalten.

- Die Menge $\mathcal{V}$ der vollständigen Graphen enthält beliebig einsame Elemente — dies war ja gerade die entscheidende Eigenschaft, die wir in Kapitel 4 ausgenutzt haben.
- Die Menge aller vollständigen Binärbäume mit Wurzel 0, bei denen die Söhne von Knoten k die Knoten $2k+1$ und $2k+2$ sind, enthält beliebig einsame Elemente, da man jeweils eine ganze „Ebene" des Baumes hinzufügen oder entfernen muß, um wieder einen vollständigen Binärbaum zu erhalten.
- Lockern wir diese Forderung auf und verlangen nur noch, daß der Graph, *unabhängig* von den Nummern der Knoten, ein vollständiger Binärbaum ist, so enthält die Menge aller solchen Binärbäume keine beliebig einsamen Graphen mehr. Man kann ja eine Kante (k, k'), stets durch eine andere Kante (k, k'') ersetzen, wenn k' keinen Nachfolger

[1] Ein Graph, der einziges Element einer Menge $\mathcal{M}$ ist, ist in $\mathcal{M}$ beliebig einsam. Diesen pathologischen Fall wollen wir ausschließen.

hat und k'' noch nicht im Baum enthalten war. Auf diese Weise erhält man zu jedem vollständigen Binärbaum einen von ihm verschiedenen im Abstand 2.[2].

- Mit der gleichen Argumentation kann man zeigen, daß die Menge der Graphen, deren Kantenzahl eine Quadratzahl ist, keine beliebig einsamen Elemente enthält.

□

Lemma 6.3 (Eigenschaften von Mengen mit einsamen Graphen)
Ist $\mathcal{M} \subseteq \mathcal{G}$ eine Menge, die beliebig einsame Graphen enthält, so hat $\mathcal{M}$ folgende Eigenschaften:

1. *$\mathcal{M}$ ist unendlich.*
2. *$\mathcal{G} \setminus \mathcal{M}$ ist unendlich.*
3. *Zu jedem $n \in \mathbb{N}$ existiert ein Graph G_n mit $|G_n| \geq n$, der n-einsam in $\mathcal{M}$ ist.*

Beweis. 1. Wir führen einen Beweis durch Kontraposition und nehmen an, $\mathcal{M}$ wäre endlich. Dann existiert

$$k \quad := \quad \max\left\{d\left(G_1, G_2\right) \mid G_1, G_2 \in \mathcal{M}\right\},$$

da die Menge, deren Maximum gebildet wird, endlich und nicht leer ist. Es gibt in $\mathcal{M}$ keinen $(k+1)$-einsamen Graphen. Denn existierte ein solcher Graph G, so gäbe es wegen der Bedingung $|\mathcal{M}| > 1$ einen Graphen $G' \in \mathcal{M}, G' \neq G$, und nach Wahl von k ist

$$d\left(G, G'\right) \quad < \quad k+1,$$

im Widerspruch zur Definition der $(k+1)$-Einsamkeit. Also enthält $\mathcal{M}$ nicht beliebig einsame Elemente.

2. Sei $G \in \mathcal{M}$ ein in $\mathcal{M}$ 1-einsamer Graph und

$$k \quad := \quad \max\left\{x \mid x \in G\right\}.$$

Dann gilt für jedes $x > k$, daß $(0, x) \notin G$, also auch $G \cup \{(0, x)\} \neq G$. Andererseits ist für jedes solche x offensichtlich

$$d\left(G, G \cup \{(0, x)\}\right) \quad = \quad 1,$$

[2]außer für den leeren Baum, aber der ist trivialerweise nicht beliebig einsam.

so daß die Definition der 1-Einsamkeit für jedes $x > k$ liefert, daß $G \cup \{(0,x)\} \notin \mathcal{M}$. Also ist

$$\{G \cup \{(0,x)\} \mid x \in \mathbb{N}, x > k\} \quad \subseteq \quad \mathcal{G} \setminus \mathcal{M},$$

insbesondere ist also $\mathcal{G} \setminus \mathcal{M}$ unendlich.

3. Seien $G, G' \in \mathcal{M}, G \neq G'$ zwei beliebige, verschiedene Graphen in $\mathcal{M}$ (ihre Existenz ist wegen $|\mathcal{M}| > 1$ gesichert). Sei weiterhin $m > d\,(G, G')$ und $k := |G|$. Dann kann G nicht m-einsam in $\mathcal{M}$ sein.

   Sei nun $n \in \mathbb{N}$ beliebig und G_n ein Graph, der $(n+m+k)$-einsam in $\mathcal{M}$ ist (ein solcher existiert nach Voraussetzung). Dann ist $G \neq G_n$ (da ja G insbesondere nicht $(n+m+k)$-einsam sein kann), und wir erhalten aus der Definition der $(n+m+k)$-Einsamkeit

   $$n+m+k \quad \leq \quad d\,(G_n, G) \quad \leq \quad |G_n| + |G| \quad = \quad |G_n| + k.$$

   Also ist $|G_n| \geq m+n \geq n$, und da die $(n+m+k)$-Einsamkeit von G_n auch die n-Einsamkeit impliziert, hat G_n alle behaupteten Eigenschaften.

■

Zur Spezifikation einer Menge $\mathcal{M}$ von Graphen wollen wir eine Signatur $\Sigma_{\mathrm{GRAPH}_{\mathcal{M}}}$ bereitstellen, die der Signatur Σ_{CLIQUE} entspricht:

signature $\Sigma_{\mathrm{GRAPH}_{\mathcal{M}}}$
import $\Sigma_{\mathrm{GRAPH}}, \Sigma_{\mathrm{BOOL}}$
functions `mgraph : Graph` $\rightarrow$ `Bool`
end signature

Wir definieren hierzu die $\Sigma_{\mathrm{GRAPH}_{\mathcal{M}}}$-Algebra $\mathbf{GRAPH}_{\mathcal{M}}$ durch

$$\begin{array}{lcl} \mathbf{GRAPH}_{\mathcal{M}}|_{\Sigma_{\mathrm{GRAPH}}} & = & \mathbf{GRAPH}, \\ \mathbf{GRAPH}_{\mathcal{M}}|_{\Sigma_{\mathrm{BOOL}}} & = & \mathbf{BOOL} \\ \mathtt{mgraph}^{\mathbf{GRAPH}_{\mathcal{M}}}(G) & = & \begin{cases} \textit{true} & \text{, falls } G \in \mathcal{M} \\ \textit{false} & \text{, sonst} \end{cases} \end{array}$$

Als Resultat über die Spezifizierbarkeit von $\mathbf{GRAPH}_{\mathcal{M}}$ können wir festhalten:

Satz 6.4 (Nicht-$\boldsymbol{QF}$-Spezifizierbarkeit von $\mathbf{GRAPH}_{\mathcal{M}}$)
Es sei $\mathcal{M} \subseteq \mathcal{G}$ eine Menge von Graphen, so daß $\mathcal{M}$ oder $\mathcal{G} \setminus \mathcal{M}$ beliebig einsame Graphen enthalten. Dann existiert keine quantorenfreie Spezifikation $SP \in QF(\Sigma_{\mathrm{GRAPH}_{\mathcal{M}}})$ mit $Mod(SP) = [\mathbf{GRAPH}_{\mathcal{M}}]$.

Beispielsweise hat die Menge $\mathcal{V}$ der vollständigen Graphen im Sinne von Kapitel 4 alle geforderten Eigenschaften, da Lemma 4.13 gerade aussagt, daß für jedes m der Graph G_m in der Menge $\mathcal{V}$ der vollständigen Graphen $(2m+1)$-einsam, also insbesondere auch m-einsam ist.

Beweis von Satz 6.4. Die Menge $\mathcal{V}$ der vollständigen Graphen enthält beliebig einsame Graphen. Mehr noch: eine genauere Betrachtung des Beweises von 4.5 zeigt, daß wir von speziellen Eigenschaften der Menge $\mathcal{V}$ im Beweis keinen Gebrauch gemacht haben. Die einzige Eigenschaft, die wir verwendet haben, war die $(m+1)$-Einsamkeit von G_m in der Menge $\mathcal{V}$ und die Mächtigkeit von G_m.

Daher liegt es nahe, den Beweis mit lediglich sehr geringfügigen Änderungen zu übertragen (bis auf die offensichtliche Änderung, das Funktionssymbol `clique` durch `mgraph` zu ersetzen).

Zu jedem $m \in \mathbb{N}$ sei $G_m \in \mathcal{M}$ ein Graph mit $|G_m| \geq m+1$, der $(m+1)$-einsam in $\mathcal{V}$ ist. Ein solcher existiert auch für jedes $m \in \mathbb{N}$ nach Lemma 6.3. Wir definieren dann unsere Nicht-Standard-Modelle $\mathbf{GRAPH}_{\mathcal{M},m}$ durch

$$\begin{array}{lcl}
\mathbf{GRAPH}_{\mathcal{M},m}|_{\Sigma_{\mathrm{GRAPH}}} & = & \mathbf{GRAPH}, \\
\mathbf{GRAPH}_{\mathcal{M},m}|_{\Sigma_{\mathrm{BOOL}}} & = & \mathbf{BOOL} \text{ und} \\
\mathtt{mgraph}^{\mathbf{GRAPH}_{\mathcal{M},m}}(G) & = & \begin{cases} \mathit{false} & \text{, für } G = G_m \\ \mathtt{mgraph}^{\mathbf{GRAPH}_{\mathcal{M}}}(G) & \text{, sonst} \end{cases}
\end{array}$$

Bei der Modifikation der Variablenbelegung wählen wir das n (aus Definition 4.17) nicht nur größer als $\max_v(\varphi)$, sondern auch größer als alle in G_m enthaltenen Knoten. Dann können wir alle anderen Definitionen und Lemmata direkt übernehmen — auf Lemma 4.13 können wir verzichten, weil seine Aussage schon aus der $(m+1)$-Einsamkeit von G_m folgt. An der Stelle im Beweis von Lemma 4.22, an der Lemma 4.13 verwendet wird, verwenden wir stattdessen die Definition der $(m+1)$-Einsamkeit.

Ebenfalls durch die Definition der $(m+1)$-Einsamkeit erhalten wir die Entsprechung zur Gleichung

$$\mathtt{clique}^{\mathbf{CLIQUE}}((t_i^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}}) \quad = \quad \mathit{false}.$$

Denn wegen $(n+1, n+2) \in (t_i^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}}$ und nach Wahl von n ist $(t_i^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} \neq G_m$, und wegen der $(m+1)$-Einsamkeit von G_m damit $(t_i^G)^{v_{\varphi,m,n},\mathbf{CLIQUE}} \notin \mathcal{M}$.

Außerdem muß noch eine Abschätzung im Beweis von Lemma 4.21 abgeändert werden, und zwar haben wir anstelle von $|G_m| = (m+1)^2$ nunmehr $|G_m| \geq m+1$, was für den Widerspruchsbeweis jedoch ausreichend ist.

Mit diesen Modifikation kann man unmittelbar nachprüfen, daß alle Beweisschritte weiterhin gültig bleiben. Damit haben wir unseren Satz für den Fall bewiesen, daß $\mathcal{M}$ selbst beliebig einsame Graphen enthält.

Enthält jedoch nur $\mathcal{G} \setminus \mathcal{M}$ beliebig einsame Graphen, so müssen die G_m als Vertreter von $\mathcal{G} \setminus \mathcal{M}$ gewählt werden, und im Beweis müssen die Rollen der Literale $\texttt{mgraph}(t^G) = \texttt{true}$ und $\texttt{mgraph}(t^G) = \texttt{false}$ vertauscht werden. Literale der Form $t_1^B = t_2^B$ bzw. $t_1^B \neq t_2^B$ lassen sich ja immer durch diese beiden Arten von Literalen ausdrücken, da $t_1^B = t_2^B$ in allen uns interessierenden Algebren äquivalent ist zu $(t_1^B = \texttt{true} \wedge t_2^B = \texttt{true}) \vee (t_1^B = \texttt{false} \wedge t_2^B = \texttt{f})$. Auf eine detaillierte Ausarbeitung wollen wir hier verzichten, da der Beweis keinerlei neue Ideen benötigt, sondern nur die veränderten Bezeichnungen. ■

Satz 6.4 macht nur eine Aussage über die Nicht-Spezifizierbarkeit der entsprechenden Datentypen $\mathbf{GRAPH}_{\mathcal{M}}$ mittels quantorenfreier Axiome. Es wird *keine* Aussage über die Existenz einer PL-Spezifikation gemacht, und tatsächlich werden viele der betrachteten Datentypen $\mathbf{GRAPH}_{\mathcal{M}}$ nicht PL-spezifizierbar sein. Etwa läßt die in [Sch97a] bewiesene Nicht-Spezifizierbarkeit der Quadrat-Funktion über der Signatur $\Sigma = \{\texttt{zero}, \texttt{succ}, \texttt{square}\}$ vermuten, daß die Eigenschaft von Graphen, vollständig zu sein und nur Quadratzahlen als Knoten zu besitzen, nicht PL-spezifizierbar ist. Ein Beweis dürfte allerdings sehr aufwendig sein, da sich die in [Sch97a] angewandten Techniken nicht unmittelbar auf dieses Beispiel übertragen lassen.

Andererseits besagt Satz 6.4 auch nicht, daß die Algebra $\mathbf{GRAPH}_{\mathcal{M}}$ für eine Menge $\mathcal{M}$, die nicht beliebig einsame Graphen enthält, QF-spezifizierbar sein muß. Beispielsweise ist vermutlich die Eigenschaft eines Graphen, zusammenhängend zu sein, nicht ohne Quantoren spezifizierbar. Entsprechend zu obigem Abschnitt liegt es auch nahe, daß die Eigenschaft eines Graphen, daß seine Mächtigkeit eine Quadratzahl ist, nicht PL-spezifizierbar, insbesondere also auch nicht QF-spezifizierbar ist. Beide Mengen enthalten jedoch keine beliebig einsamen Graphen im Sinne unsere Definition (vgl. auch Beispiel 6.2).

6.2 Endliche und frei erzeugte Datentypen

Während wir im letzten Abschnitt eine Klasse von Datentypen kennengelernt haben, die alle nicht quantorenfrei spezifiziert werden können, wollen wir uns hier zwei Klassen zuwenden, deren sämtliche Vertreter quantorenfrei spezifizierbar sind.

Die erste Klasse ist die der endlichen Datentypen, d.h. alle Trägermengen sind endlich. Somit sind insbesondere die Definitionsbereiche aller Funktionen des Datentyps endlich, und jede Funktion kann einfach durch eine endliche Menge von Grundgleichungen spezifiziert werden, die zu allen Eingangswerten den Funktionswert festlegen, sowie eine endliche Menge von Ungleichungen für je zwei verschiedene Trägerelemente.

Die zweite Klasse ist die der *frei erzeugten Datentypen* (vgl. Definition 2.24). Aus der Definition der freien Erzeugtheit eines Datentyps folgt, daß alle Funktionen des Datentyps injektiv sein müssen, und die Bilder verschiedener Funktionen mit gleicher Zielsorte ebenfalls verschieden sind. Gleichheiten gelten nicht — außer den trivialen Gleichheiten $x = x$.

Zur Spezifikation benötigt man also für jede Funktion f der Stelligkeit k ein Axiom der Form

$$(\mathtt{x}_1 \neq \mathtt{y}_1 \vee \ldots \vee \mathtt{x}_k \neq \mathtt{y}_k) \quad \rightarrow \quad f(\mathtt{x}_1, \ldots, \mathtt{x}_k) \neq f(\mathtt{y}_1, \ldots, \mathtt{y}_k)$$

und für je zwei Funktionen f, g der Stelligkeit k, j mit der gleichen Zielsorte ein Axiom der Form

$$f(\mathtt{x}_1, \ldots, \mathtt{x}_k) \quad \neq \quad g(\mathtt{y}_1, \ldots, \mathtt{y}_j).$$

Zwei Beispiele für quantorenfreie Spezifikationen frei erzeugter Datentypen sind die Spezifikationen NAT und BOOL aus Abschnitt 2.2.3.

Als Verallgemeinerung der frei erzeugten Datentypen werden in [Sch98] die fast frei erzeugten Datentypen vorgestellt. Während bei frei erzeugten Datentypen jedes Trägerelement nur von genau einer Funktion und auch nur genau einmal angenommen wird, dürfen bei fast frei erzeugten Datentypen endlich viele Elemente mehrfach angenommen werden, allerdings auch nur jeweils endlich oft.

Damit enthalten die fast frei erzeugten Datentypen insbesondere die frei erzeugten und die endlichen Datentypen. Auch die fast frei erzeugten Datentypen sind QF-spezifizierbar. Auf den (nicht wesentlich komplizierteren) Beweis wollen wir an dieser Stelle allerdings verzichten und verweisen stattdessen auf [Sch98].

6.3 Komplexität

In den letzten beiden Abschnitten haben wir (vergleichsweise kleine) Klassen von Datentypen kennengelernt, für die wir die QF-Spezifizierbarkeit bzw. Nicht-QF-Spezifizierbarkeit nachweisen konnten. Ein besonders erfolgversprechender Ansatz zur Identifikation eines hinreichenden oder notwendigen Kriteriums für QF-Spezifizierbarkeit ist der, die Komplexität des betrachteten Datentyps bezüglich eines geeigneten Komplexitätsmaßes zu wählen.

6.3.1 Berechnungskomplexität von Funktionen

Ein ähnlicher Ansatz wird beispielsweise in der endlichen Modelltheorie verfolgt (vgl. etwa [EF95]), wo, grob gesagt, untersucht wird, welche relationalen Strukturen (eine oder mehrere Relationen auf Grundmengen) sich mit Hilfe prädikatenlogischer Formeln charakterisieren lassen. Dort ergeben sich enge Zusammenhänge zwischen der benötigten Anzahl Formeln, Variablen und Quantoren einerseits und der Berechnungskomplexität der Prädikate andererseits (vgl. etwa [Imm82], [Imm89]).

Allerdings kommen in den entsprechenden Formeln weder Variablen für die betrachteten relationalen Strukturen (ein Spezialfall solcher Strukturen wären gerade unsere Graphen aus Kapitel 4 oder die k-Graphen aus Kapitel 5) noch Signatursymbole für die zu charakterisierenden Eigenschaften vor (in unserem Fall etwa `clique`). Damit ist die Ausdrucksmächtigkeit natürlich eingeschränkt — zum Beispiel ist eine induktive Spezifikation von Eigenschaften nicht möglich. So läßt sich zum Beispiel auch keine prädikatenlogische Formel angeben, die genau für alle zusammenhängenden Graphen gilt, während eine Spezifikation mit Hilfe von Quantoren über Graphen sehr wohl möglich ist.

Somit lassen sich zumindest nicht alle Ergebnisse aus der endlichen Modelltheorie auf die von uns betrachtete Spezifikation monomorpher Datentypen übertragen, und tatsächlich werden wir zeigen können, daß die Berechnungskomplexität von Funktionen im Sinne der nachfolgenden Definition *kein* geeignetes Kriterium für die quantorenfreie Spezifizierbarkeit ist.

Um uns in unserem Ergebnis nicht auf ein Berechnungsmodell festzulegen, wollen wir zunächst den Begriff des abstrakten Komplexitätsmaßes (vgl. etwa [Pau78], [Bör89], [Kum92]) einführen:

Definition 6.5 (Abstraktes Komplexitätsmaß (Blum, 1967))
Sei φ eine Gödelnumerierung der berechenbaren Funktionen von $\mathbb{N}$ nach $\mathbb{N}$. Eine Funktion $T \in P_2$ heißt **Komplexitätsmaß**, falls sie folgende zwei Eigenschaften erfüllt:

1. Für alle $i, j \in \mathbb{N}$ gilt:
$$\varphi_i(j)\downarrow \iff T(i,j)\downarrow$$
2. $graph(T)$ ist entscheidbar.

$\square$

Ein Komplexitätsmaß T ordnet jeder *terminierenden* Programmausführung des Programms i auf Eingabe j eine Zahl $T(i,j)$ als Komplexität der Berechnung zu. Durch die zweite Bedingung an T ist sichergestellt, daß sich die Komplexität tatsächlich bestimmen läßt.

Ein Beispiel für ein solches Komplexitätsmaß wäre die Zeitkomplexität T_{Time} auf einer Turingmaschine. Daß das erste Axiom erfüllt ist, ist offensichtlich. Um zu entscheiden, ob $(i,j,w) \in graph(T_{Time})$, kann man einfach w Schritte der Berechnung von Turingmaschine i auf Eingabe j simulieren und überprüfen, ob die Maschine nach genau w Schritten terminiert. In diesem Fall ist $(i,j,w) \in graph(T_{Time})$, andernfalls nicht. Eine ähnliche Argumentation kann man für andere verbreitete Komplexitätsmaße durchführen.

Die für uns entscheidende Eigenschaft von Komplexitätsmaßen im obigen Sinn ist die Existenz beliebig komplexer Funktionen auf den natürlichen Zahlen. Sie wird ausgedrückt durch folgenden Satz (vgl. z.B. [Kum92]):

Satz 6.6 (Satz von Rabin, 1960)
Sei $k \in R_1$ (eine Komplexitätsschranke) und T ein Komplexitätsmaß. Dann existiert eine Menge $A \subseteq \mathbb{N}$ mit charakteristischer Funktion χ_A, so daß für alle i gilt:

$$\varphi_i = \chi_A \implies T(i,j) > k(j) \text{ für alle außer endlich vielen } j.$$

Ein Beweis dieses Satzes ist in Standardlehrbüchern zur Komplexitätstheorie wie [Pau78], [Bör89] oder auch in [Kum92] zu finden. Zur charakteristischen Funktion $f := \chi_A$ der Menge A können wir den Datentypen $\mathbf{NAT}_f$ im Sinne von Kapitel 3 betrachten. Dieser ist nach Satz 3.1 QF-spezifizierbar, d.h. wir erhalten zu jeder Komplexitätsschranke k einen QF-spezifizierbaren Datentypen $\mathbf{NAT}_f$, der eine Funktion mit Berechnungskomplexität mindestens k enthält und trotzdem QF-spezifizierbar ist.

Somit erhalten wir für die Berechnungskomplexität von Funktionen eines Datentyps im Sinne von Blum keine notwendige Bedingung für QF-Spezifizierbarkeit, da Datentypen mit beliebig komplexen Funktionen QF-spezifizierbar sind — allerdings haben wir auch keine Bedingungen an die Komplexität der QF-Spezifikation gemacht.

6.3.2 Komplexität von Datentypen

In der Literatur gibt es Ansätze, Komplexitätsmaße für Datentypen zu definieren (vgl. etwa [EM81], [AT82]). Wir wollen uns hier kurz mit dem Ansatz von H. Ehrig und B. Mahr befassen, ohne technische Details zu präsentieren — für exakte Definitionen siehe [EM81].

Dort wird die Komplexität der Implementierung eines abstrakten Datentyps durch einen anderen abstrakten Datentyp definiert, d.h. es handelt sich um ein relatives Komplexitätsmaß.

Zur Implementierung eines Datentyps wird die Signatur um Hilfssorten und Hilfsfunktionen, und die Axiomenmenge (die als Menge von Gleichungen vorausgesetzt wird) um weitere Gleichungen angereichert. Eine Berechnung ist eine Folge von Anwendungen von Gleichungen, die einen Term der implementierten Sorte in einen Term der implementierenden überführt. Die Komplexität einer Berechnung ist die Anzahl der „Anwendungen“ von Gleichungen in einer kürzesten Berechnung.

Bezugnehmend auf das Beispiel $\mathbf{NAT}_f$ (Kapitel 3) wollen wir die Komplexität der Implementierung von Datentypen $\mathbf{NAT}_f$ durch **NAT-MULT** untersuchen. Wir werden nachweisen, daß es auch bezüglich dieser Definition von Komplexität (QF-spezifizierbare) Datentypen $\mathbf{NAT}_f$ gibt, die beliebig komplexe Implementierungen erfordern.

Zum Beweis wollen wir wiederum den Satz von Rabin verwenden und müssen also ein Blumsches Komplexitätsmaß auf Implementierungen definieren. Dazu ordnen wir einer Implementierung $IMPL$ (mit einer Menge E von Gleichungen) über der Signatur $\Sigma_{\mathrm{NAT_f}}$ auf naheliegende Weise eine partielle Funktion $f_{IMPL} : \mathbb{N} \to \mathbb{N}$ zu durch:

$$f_{IMPL}(n) \quad := \quad \begin{cases} m & \text{, falls } E \models \mathtt{f}(\mathtt{succ}^n(\mathtt{zero})) = \mathtt{succ}^m(\mathtt{zero}) \\ \uparrow & \text{, falls kein solches } m \text{ existiert .} \end{cases}$$

Ist die Gleichungsmenge E konsistent, so ist f_{IMPL} wohldefiniert.

Da sich unter Verwendung von Hilfsfunktionen und Hilfssorten jede berechenbare Funktion durch Gleichungen spezifizieren läßt (s. [BT82], [BT87]), definieren wir auf diese Weise eine bijektive Abbildung zwischen den Implementierungen über der Signatur $\Sigma_{\mathrm{NAT_f}}$ und den partiellen Funktionen auf den natürlichen Zahlen.

Mit Hilfe der Zuordnung einer Funktion f_{IMPL} zu einer Implementierung $IMPL$ erhalten wir auf naheliegende Weise eine Gödelnumerierung der partiellen Funktionen $f : \mathbb{N} \to \mathbb{N}$, da jede Implementierung nur endlich viele Hilfssymbole und Gleichungen hinzufügt, und sich demnach alle Implementierungen effektiv und bijektiv auf die natürlichen Zahlen abbilden lassen. Sei etwa g eine solche bijektive Abbildung von den natürlichen Zahlen auf die Implementierungen über der Signatur $\Sigma_{\mathrm{NAT_f}}$. Dann ist

$$\psi(i,j) \quad := \quad f_{f(i)}(j)$$

eine Gödelnumerierung der partiellen Funktionen $f : \mathbb{N} \to \mathbb{N}$.

Wir definieren im folgenden eine Funktion $T : \mathbb{N}^2 \to \mathbb{N}$ und zeigen, daß diese ein Komplexitätsmaß im Sinne von Definition 6.5 ist. Wir setzen $T(i,j) := w$, falls $\psi(i,j) \downarrow$ und w die minimale Anzahl von Termen t_n ist mit

$$E \quad \models \quad \mathtt{f}(\mathtt{succ}^j(\mathtt{zero})) = t_1 = \ldots = t_{w-1} = \mathtt{succ}^{\psi_i(j)}(\mathtt{zero}).$$

Falls $\psi(i,j) \uparrow$, so setzen wir $T(i,j) \uparrow$. Dann ist klar, daß T die erste Bedingung an ein Komplexitätsmaß erfüllt.

Um zu verifizieren, daß T auch die zweite Bedingung erfüllt, geben wir ein Entscheidungsverfahren für $graph(T)$ an. Da ein Term nur endlich viele Subterme hat und die Menge E der Gleichungen ebenfalls endlich ist, kann zu jedem Zeitpunkt der Berechnung nur eine endliche Auswahl von Berechnungsschritten vorgenommen werden, also gibt es zu jedem w auch nur endlich viele Berechnungen der Länge w.

Um zu entscheiden, ob $(i,j,w) \in graph(T)$, simuliert man also alle möglichen Berechnungen mit Länge w der Implementierung i auf Eingabe j. Falls eine Berechnung in genau w Schritten $\mathtt{f}(\mathtt{succ}^j(\mathtt{zero})) = \mathtt{succ}^m(\mathtt{zero})$ für ein $m \in \mathbb{N}$ zeigt, so prüft man (ebenfalls durch Simulation), ob es eine kürzere Berechnung gibt. Gibt es eine solche nicht, ist $(i,j,w) \in graph(T)$. Falls die Gleichung nicht durch w Schritte gefolgert werden kann oder eine Ableitung mit weniger als w Schritten existiert, ist $(i,j,w) \notin graph(T)$. Unter Verwendung der Churchschen These folgt aus dem informell angegebenen Algorithmus, daß $graph(T)$ entscheidbar ist.

Also ist T Komplexitätsmaß, und wir können aus dem Satz von Rabin schließen, daß es bezüglich dieses Komplexitätsmaßes beliebig komplexe Probleme gibt. Dies impliziert aber unmittelbar, daß die Komplexität der entsprechenden Datentypen im Sinne von [EM81] beliebig groß wird. Also ist auch die Komplexität von Implementierungen abstrakter Datentypen im Sinne von [EM81] kein geeignetes Hilfsmittel zur Charakterisierung der QF-Spezifizierbarkeit.

6.4 Abgeschlossenheit gegen Unteralgebren

Die Fragestellung, wann man zu einer Formel eine äquivalente Formel ohne Quantoren angeben kann, ist auch ein Forschungsgebiet der Modelltheorie (vgl. etwa [Rob74]). Dort wird gezeigt, daß algebraische Abschlußeigenschaften der Klasse der Modelle einer Formel hinreichend für die Existenz einer äquivalenten quantorenfreien Formel sind. Dort werden die Modelle jedoch nicht auf die termerzeugten eingeschränkt, so daß die Ergebnisse in unserem Fall nur bedingt anwendbar sind. Für den Rest dieses Abschnitts meinen wir mit Modell immer ein beliebiges (d.h. nicht notwendig termerzeugtes) Modell. Wenn wir Termerzeugtheit fordern, wird dies explizit erwähnt.

Eine Unteralgebra A einer Σ-Algebra B ist eine unter den Funktionen von B abgeschlossene Σ-Algebra, deren Träger Teilmengen der Träger von B sind.

Definition 6.7 (Unteralgebren)
Sei $\Sigma = (S, F)$ Signatur und A, B Σ-Algebren. A heißt **Unteralgebra von B** (oder **Subalgebra**), falls gilt:

- $s^A \subseteq s^B$ für jedes $s \in S$.
- $f^A(a_1, \ldots, a_k) = f^B(a_1, \ldots, a_k)$ für jedes $f \in F$ und $a_j \in s_j^A$.

□

Definition 6.8 (Abschluß gegen Unteralgebren)
Sei Σ eine Signatur. Eine Menge $\mathcal{A}$ von Σ-Algebren heißt **abgeschlossen gegen Unteralgebren**, falls mit $A \in \mathcal{A}$ auch für jede Unteralgebra B von A gilt: $B \in \mathcal{A}$. □

Abschluß gegen Unteralgebren ist ein geeignetes Kriterium zum Nachweis der Existenz äquivalenter quantorenfreier Formeln ([Rob74, Theorem 3.3.4], dort allerdings nicht explizit für mehrsortige Algebren).

Satz 6.9 (äquivalente quantorenfreie Formeln)
Sei Σ eine Signatur und $\varphi \in PL(\Sigma)$ eine Formel. Die Modelle von φ sind genau dann gegen Unteralgebren abgeschlossen, falls eine quantorenfreie Formel $\phi \in QF(\Sigma)$ über der selben Signatur existiert, die die gleichen Modelle wie φ besitzt.

Beweis. Während die Implikation von rechts nach links vergleichsweise einfach ist, ist der Nachweis der anderen Implikation deutlich aufwendiger. Ein Beweis wird zum Beispiel in [Rob74] geführt. ■

Als einfaches Korollar erhält man

Korollar 6.10 (Quantorenfreie Spezifikation von Datentypen)
Ist A ein PL-spezifizierbarer Datentyp und $SP = (\Sigma, \Phi, F_c)$ *eine PL-Spezifikation von A, so daß die Menge* aller *Modelle von* Φ *gegen Unteralgebren abgeschlossen ist, so existiert eine quantorenfreie monomorphe Spezifikation von A.*

Für termerzeugte Datentypen haben wir nur noch die eine Richtung des Satzes 6.9 zur Verfügung, und können somit den Satz insbesondere nicht verwenden, um die Nicht-Existenz einer quantorenfreien Spezifikation eines Datentyps zu beweisen. Denn man kann quantorenfrei spezifizierbare Datentypen und zu diesen eine Spezifikation SP mit Quantoren angeben und anschließend die Menge *aller* Modelle von SP betrachten. In vielen Fällen wird diese nicht gegen Subalgebren abgeschlossen sein. Trotzdem existiert nach Wahl des Datentyps eine quantorenfreie Spezifikation, die die gleichen *termerzeugten* Modelle besitzt.

Das Kriterium wird eher angewendet, um die quantorenfreie Spezifizierbarkeit gewisser algebraischer Strukturen (wie Körper, Ringe, etc.) zu untersuchen. Bei diesen möchte man nicht unbedingt Sorten (bzw. Trägermengen) eindeutig spezifizieren, sondern nur gewisse Eigenschaften von Funktionen oder Prädikaten garantieren. Zum Nachweis der Abschlußeigenschaften hat man umgekehrt meist mächtige Hilfsmittel der Algebra zur Verfügung. Entsprechend findet sich beispielsweise in [Sho67] ein noch mehr auf Spezifikationen algebraischer Strukturen zugeschnittenes Kriterium an die Modellklasse einer Formel ([Sho67, QUANTIFIER ELIMINATION THEOREM, S.85]).

6.5 Initiale Spezifikation

In den bisherigen Kapiteln dieser Arbeit wurden zwei weit verbreitete Methoden zur Spezifikation abstrakter Datentypen mit *loser* Semantik betrachtet. In diesem Abschnitt wollen wir den Rahmen der losen Spezifikation verlassen und uns zwei weiteren verbreiteten Spezifikationstechniken zuwenden, die sich initialer Semantik bedienen. Wir wollen untersuchen, ob die Spezifizierbarkeit mit diesen Logiken unter initialer Semantik geeignet ist, Aussagen über quantorenfreie Spezifizierbarkeit zu machen.

Wir werden — wie schon bei den in Abschnitt 6.3 betrachteten Komplexitätsmaßen — auch hier eine negative Antwort erhalten, d.h. Spezifizierbarkeit mit initialer Semantik ist nicht geeignet zur Charakterisierung der Spezifizierbarkeit mit loser Semantik.

In Abschnitt 6.5.1 wollen wir die benötigten Grundbegriffe vorstellen. Die beiden darauffolgenden Abschnitte werden dann den Ergebnissen über den Zusammenhang (bzw. Nicht-Zusammenhang) zwischen initialer und loser Spezifizierbarkeit gewidmet sein.

6.5.1 Begriffe und Definitionen

Die nachfolgenden Begriffe und Definitionen sind in dieser Form wiederum weitgehend dem Grundlagen-Kapitel aus [Sch98] entnommen. Wir definieren die Syntax der (bedingten) Gleichungslogik und die Semantik entsprechender Spezifikationen.

Definition 6.11 (Syntax der Gleichungslogik, $\boldsymbol{EL(\Sigma)}$, $\boldsymbol{CEL(\Sigma)}$)
Zu einer Signatur $\Sigma = (S, F)$ (mit zugehöriger Variablenfamilie X) sind die Formeln der **Gleichungslogik** $EL(\Sigma)$ und der **bedingten Gleichungslogik** $CEL(\Sigma)$ erklärt durch

$$\begin{array}{lcl} EL(\Sigma) & := & \left\{(t_1 = t_2) \,\middle|\, t_1, t_2 \in T(\Sigma(X))_s,\ s \in S\right\} \\ CEL(\Sigma) & := & EL(\Sigma) \cup \left\{eq_1 \wedge \cdots \wedge eq_k \to eq_{k+1} \,\middle|\, k > 0,\ eq_i \in EL(\Sigma)\right\}. \end{array}$$

Die Definition einer EL-Spezifikation (bzw. CEL-Spezifikation) zur Signatur Σ ist analog zu der einer QF-Spezifikation, wobei für die Menge Φ der Axiome in diesem Fall $\Phi \subseteq EL(\Sigma)$ (bzw. $\Phi \subseteq CEL(\Sigma)$) gilt. □

Definition 6.12 (Initiale Semantik)
Sei $SP = (\Sigma, \Phi)$ eine Spezifikation und $A \in Gen(\Sigma)$ ein termerzeugter Datentyp. A heißt initial in $Mod(SP)$, wenn zu jedem Modell $B \in Mod(SP)$ ein Σ-Homomorphismus $h : A \to B$ existiert. Wir verwenden die Bezeichnung
$I(SP) := \{A \in Gen(\Sigma, F_c) \mid A \text{ ist initial in } Mod(SP)\}$ □

Das initiale Modell einer Spezifikation SP zeichnet sich — anschaulich gesprochen — dadurch aus, daß in ihm *genau* die aus SP folgerbaren Grundgleichungen gelten, d.h. für alle Grundterme t_1, t_2:

$$A \models t_1 = t_2 \iff SP \models t_1 = t_2.$$

Analog zur QF-Spezifizierbarkeit (bzw. PL-Spezifizierbarkeit) eines Datentyps definieren wir nun den Begriff der EL-Spezifizierbarkeit und CEL-Spezifizierbarkeit.

Definition 6.13 (initiale Spezifikationsmethoden)
Ein abstrakter Datentyp $\mathcal{C}$ heißt[3]

- EL-**spezifizierbar**, falls eine EL-Spezifikation SP mit $I(SP) = \mathcal{C}$ existiert.
- CEL-**spezifizierbar**, falls eine CEL-Spezifikation SP mit $I(SP) = \mathcal{C}$ existiert.

$\square$

Man kann die Initialität auch mit Hilfe von Homomorphismen charakterisieren (häufig wird dies als die Definition eines initialen Modells verwendet).

Anhand der Definitionen ist offensichtlich, daß jeder EL-spezifizierbare Datentyp auch CEL-spezifizierbar ist. Daß die Inklusion tatsächlich echt ist, wurde in [TWW82] nachgewiesen, ein Beweis findet sich auch in [Sch98]. Dort wird eine umfassende Untersuchung der Spezifikation mittels QF, PL, EL und CEL durchgeführt und gezeigt, daß außer den offensichtlichen Inklusionen $QF \subseteq PL$ und $EL \subseteq CEL$ keine weiteren gelten.

Insbesondere wird dort und in [Sch97b] gezeigt, daß es EL-spezifizierbare Datentypen (zum Beispiel die ganzen Zahlen mit Null, Vorgänger und Nachfolger) gibt, die nicht PL-spezifizierbar sind. Auch die in den nachfolgenden beiden Abschnitten angegebenen Beispiele werden dort mitsamt ausführlichen Beweisen behandelt, so daß wir im folgenden auf Beweise verzichten wollen.

Während in [Sch98] die initiale Spezifizierbarkeit als solche von Interesse ist, betrachten wir die Fragestellung hier in einem etwas anderen Zusammenhang, der uns auch schon in den bisherigen Abschnitten dieses Kapitels beschäftigte. Nachdem das zentrale Ergebnis dieser Arbeit war, daß unter Verwendung von Quantoren tatsächlich mehr Datentypen monomorph spezifiziert werden können als ohne Quantoren, sind wir interessiert an einer möglichst genauen Charakterisierung der quantorenfreien Spezifizierbarkeit — und zwar sowohl durch positive als auch durch negative Resultate.

Im günstigsten Fall könnte man hoffen, daß die Spezifizierbarkeit eines Datentyps sowohl mit PL als auch mit CEL die Spezifikation mit quantorenfreien Axiomen sicherstellen würde. Immerhin ist ja jedes CEL-Axiom insbesondere auch ein QF-Axiom, so daß alle Gleichheiten spezifiziert werden können. Allerdings sind die Ungleichheiten bei CEL-Spezifikationen implizit

[3] Man schränkt sich bei initialer Spezifikation im allgemeinen auf Gleichungen oder bedingte Gleichungen ein, da man somit die Existenz eines Modells sicherstellt.

dadurch festgelegt, daß die entsprechende Gleichheit nicht aus der Spezifikation folgerbar ist. Bei entsprechend gearteten Beispielen kann die Definition der Ungleichheiten deutlich aufwendiger sein, und tatsächlich werden wir in den folgenden zwei Abschnitten Datentypen kennenlernen, die sowohl CEL- (bzw. EL-)spezifizierbar als auch PL-spezifizierbar, nicht jedoch QF-spezifizierbar sind.

6.5.2 Bedingte Gleichungslogik

In diesem Abschnitt werden wir den Datentyp **CLIQUE** aus Kapitel 4 so modifizieren, daß das Resultat zwar PL- und CEL- spezifizierbar, aber weder QF- noch EL-spezifizierbar ist. Hierzu nutzen wir die Beobachtung, daß für die Unvollständigkeit eines Graphen eine lokale Eigenschaft hinreichend ist — im Gegensatz zur Vollständigkeit, die eine globale Eigenschaft ist (vgl. Abschnitt 4.3).

Wenn wir also die initiale Semantik ausnutzen können, um lediglich die *unvollständigen* Graphen spezifizieren zu müssen (und die Vollständigkeit der anderen in Form von Ungleichungen zu erhalten), hätten wir einen geeigneten Datentyp konstruiert. Die Idee hierzu ist, eine neue Sorte hinzuzunehmen, die termerzeugt durch eine einzige Funktion, eine sogenannte **Reduktionsfunktion** ist.[4] Durch die Axiome an die Reduktionsfunktion fordert man, daß alle unvollständigen Graphen auf das *selbe* Element abgebildet werden, während die Bilder aller vollständigen Graphen paarweise verschieden (und insbesondere verschieden vom Bild unvollständiger Graphen) sind.

Letztere Eigenschaft kann man mit Hilfe voller Prädikatenlogik axiomatisieren, und unter initialer Semantik braucht man dafür gar keine Axiome, wenn nur keine Gleichheit zwischen den entsprechenden Elementen folgerbar ist. Wir definieren also eine neue Signatur

signature Σ_{C}		
import	Σ_{GRAPH}	
sorts	`Bool`, `Cliques`	
functions	`noedge`	: `Nat` $\times$ `Nat` $\times$ `Graph` $\to$ `Bool`
	`red`	: `Graph` $\to$ `Cliques`
end signature		

[4]Die Verwendung einer solchen Reduktionsfunktion für Prädikate anstelle einer Funktion mit dem Wertebereich `Bool` ist eine übliche Technik zur Konstruktion von Datentypen, die zwar unter initialer, nicht aber unter loser Semantik spezifizierbar sind, vgl. etwa [TWW82] oder [Sch98].

Die Funktion `noedge` benötigen wir dabei, um auszudrücken, daß zwischen zwei Knoten eines Graphen *keine* Verbindung existiert. Hierzu würden wir andernfalls eine *Ungleichung* benötigen, die ja in CEL-Axiomen nicht auftauchen darf.

Wir bezeichnen mit $\mathcal{V}$ wiederum die Menge aller vollständigen Graphen, und können dann die Algebra **CLIQUE-C** definieren:

$$
\begin{array}{lcl}
\mathbf{CLIQUE\text{-}C}|_{\Sigma_{\mathrm{GRAPH}}} & = & \mathbf{GRAPH}, \\
\mathbf{CLIQUE\text{-}C}|_{\Sigma_{\mathrm{BOOL}}} & = & \mathbf{BOOL}, \\
\mathtt{Cliques}^{\mathbf{CLIQUE\text{-}C}} & = & \mathcal{V} \cup \{\{(0,1)\}\} \\
\mathtt{noedge}^{\mathbf{CLIQUE\text{-}C}}(x,y,G) & = & \begin{cases} \mathit{true} & \text{, falls } (x,y) \notin G \\ \mathit{false} & \text{, sonst} \end{cases} \\
\mathtt{red}^{\mathbf{CLIQUE\text{-}C}}(G) & = & \begin{cases} G & \text{, falls } G \in \mathcal{V} \\ \{(0,1)\} & \text{, sonst} \end{cases}
\end{array}
$$

Wir können nun eine CEL-Spezifikation CLIQUE-C-I des so definierten Datentyps **CLIQUE-C** angeben, wobei wir noch die Abkürzung

$$\mathit{incompl} \;:\equiv\; \mathtt{connect}(\mathtt{zero}, \mathtt{succ}(\mathtt{zero}), \mathtt{empty})$$

vereinbaren wollen. Die Spezifikation werden wir in zwei Teilspezifikationen zerlegen, um sie nicht gar zu unübersichtlich werden zu lassen. Im ersten Teil — der Spezifikation GRAPH-C-I, wird der Datentyp für Graphen mit der zusätzlich benötigten Funktion `noedge` definiert. Der zweite Teil (CLIQUE-C-I) fügt dann die Definition der Reduktionsfunktion hinzu.

specification GRAPH-C-I

sorts $\mathtt{Bool}, \mathtt{Nat}, \mathtt{Graph}$

functions

$$
\begin{array}{lll}
\mathtt{true}, \mathtt{false} & : & \to \mathtt{Bool} \\
\mathtt{zero} & : & \to \mathtt{Nat} \\
\mathtt{succ} & : \mathtt{Nat} & \to \mathtt{Nat} \\
\mathtt{empty} & : & \to \mathtt{Graph} \\
\mathtt{connect} & : \mathtt{Nat} \times \mathtt{Nat} \times \mathtt{Graph} & \to \mathtt{Graph} \\
\mathtt{noedge} & : \mathtt{Nat} \times \mathtt{Nat} \times \mathtt{Graph} & \to \mathtt{Bool}
\end{array}
$$

variables

$$
\begin{array}{ll}
\mathtt{x}, \mathtt{y}, \mathtt{u}, \mathtt{v} & : \mathtt{Nat} \\
\mathtt{g}, \mathtt{h} & : \mathtt{Graph}
\end{array}
$$

axioms

(1) $\mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g})) = \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g})$

(2) $\mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{connect}(\mathtt{u}, \mathtt{v}, \mathtt{g}))$
$\quad = \mathtt{connect}(\mathtt{u}, \mathtt{v}, \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g}))$

(3) $\mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g}) = \mathtt{g} \to \mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{g}) = \mathtt{false}$

(4) $\mathtt{noedge}(\mathtt{zero}, \mathtt{y}, \mathtt{connect}(\mathtt{succ}(\mathtt{u}), \mathtt{v}, \mathtt{empty})) = \mathtt{true}$

(5) $\mathtt{noedge}(\mathtt{x}, \mathtt{zero}, \mathtt{connect}(\mathtt{u}, \mathtt{succ}(\mathtt{v}), \mathtt{empty})) = \mathtt{true}$

(6) $\mathtt{noedge}(\mathtt{succ}(\mathtt{x}), \mathtt{y}, \mathtt{connect}(\mathtt{zero}, \mathtt{v}, \mathtt{empty})) = \mathtt{true}$

(7) $\mathtt{noedge}(\mathtt{x}, \mathtt{succ}(\mathtt{y}), \mathtt{connect}(\mathtt{u}, \mathtt{zero}, \mathtt{empty})) = \mathtt{true}$

(8) $\mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{connect}(\mathtt{u}, \mathtt{v}, \mathtt{empty})) = \mathtt{true}$
$\quad \to \mathtt{noedge}(\mathtt{succ}(\mathtt{x}), \mathtt{y}, \mathtt{connect}(\mathtt{succ}(\mathtt{u}), \mathtt{v}, \mathtt{empty}))$
$\qquad = \mathtt{true}$

(9) $\mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{connect}(\mathtt{u}, \mathtt{v}, \mathtt{empty})) = \mathtt{true}$
$\quad \to \mathtt{noedge}(\mathtt{x}, \mathtt{succ}(\mathtt{y}), \mathtt{connect}(\mathtt{u}, \mathtt{succ}(\mathtt{v}), \mathtt{empty}))$
$\qquad = \mathtt{true}$

(10) $\mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{empty}) = \mathtt{true}$

(11) $\mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{g}) = \mathtt{true}$
$\quad \wedge \mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{connect}(\mathtt{u}, \mathtt{v}, \mathtt{empty})) = \mathtt{true}$
$\to \mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{connect}(\mathtt{u}, \mathtt{v}, \mathtt{g})) = \mathtt{true}$

end specification

specification CLIQUE-C-I

import GRAPH-C-I

sorts $\mathtt{Cliques}$

functions $\mathtt{red} : \mathtt{Graph} \to \mathtt{Cliques}$

variables $\mathtt{x}, \mathtt{y}, \mathtt{u}, \mathtt{v} : \mathtt{Nat}$
$\mathtt{g}, \mathtt{h} : \mathtt{Graph}$

axioms

$$
\begin{array}{ll}
(12) & \mathtt{connect}(\mathtt{x}, \mathtt{u}, \mathtt{g}) = \mathtt{g} \land \mathtt{connect}(\mathtt{y}, \mathtt{v}, \mathtt{g}) = \mathtt{g} \\
& \quad \land\, \mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{g}) = \mathtt{true} \\
& \to \mathtt{red}(\mathtt{g}) = \mathtt{red}(\mathit{incompl}) \\
(13) & \mathtt{connect}(\mathtt{x}, \mathtt{u}, \mathtt{g}) = \mathtt{g} \land \mathtt{connect}(\mathtt{v}, \mathtt{y}, \mathtt{g}) = \mathtt{g} \\
& \quad \land\, \mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{g}) = \mathtt{true} \\
& \to \mathtt{red}(\mathtt{g}) = \mathtt{red}(\mathit{incompl}) \\
(14) & \mathtt{connect}(\mathtt{u}, \mathtt{x}, \mathtt{g}) = \mathtt{g} \land \mathtt{connect}(\mathtt{y}, \mathtt{v}, \mathtt{g}) = \mathtt{g} \\
& \quad \land\, \mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{g}) = \mathtt{true} \\
& \to \mathtt{red}(\mathtt{g}) = \mathtt{red}(\mathit{incompl}) \\
(15) & \mathtt{connect}(\mathtt{u}, \mathtt{x}, \mathtt{g}) = \mathtt{g} \land \mathtt{connect}(\mathtt{v}, \mathtt{y}, \mathtt{g}) = \mathtt{g} \\
& \quad \land\, \mathtt{noedge}(\mathtt{x}, \mathtt{y}, \mathtt{g}) = \mathtt{true} \\
& \to \mathtt{red}(\mathtt{g}) = \mathtt{red}(\mathit{incompl})
\end{array}
$$

end specification

Die Spezifikation wird sehr viel umfangreicher dadurch, daß wir die Nichtexistenz einer Kante umständlich mit Hilfe der Funktion $\mathtt{noedge}$ kodieren müssen. Die Axiome (4)–(9) definieren die Ungleichheit zweier Kanten, in den Axiomen (10)–(11) wird diese verwendet, um die Nichtexistenz von Kanten in Graphen zu spezifizieren. Die Tatsache, daß nun gleich vier Axiome (die Axiome (12)–(15)) zur Spezifikation von $\mathtt{red}$ vonnöten sind, liegt daran, daß die verschiedenen Fälle der Disjunktion aus der Spezifikation CLIQUE aufgelöst und in getrennten Axiomen formuliert werden mußten.

Ein Beweis dafür, daß CLIQUE-C-I unter initialer Semantik tatsächlich den Datentyp **CLIQUE-C** spezifiziert, wird in [Sch98] gegeben. Eine Spezifikation von **CLIQUE-C** unter loser Semantik sieht sehr ähnlich aus wie die Spezifikation CLIQUE aus Kapitel 4. Zur Spezifikation von $\mathtt{noedge}$ nimmt man die Axiome (3)–(9) der Spezifikation CLIQUE-C-I hinzu. Außerdem ersetzt man im Axiom (4) der Spezifikation CLIQUE das Literal $\mathtt{clique}(\mathtt{g}) = \mathtt{true}$ durch

$$\mathtt{red}(\mathtt{g}) \;\neq\; \mathtt{red}(\mathit{incompl}).$$

Schließlich benötigt man noch ein weiteres Axiom, um zu spezifizieren, daß die Bilder aller vollständiger Graphen paarweise verschieden sind. Ein solches Axiom ist etwa das folgende:

$$(\texttt{red}(\texttt{g}) \neq \texttt{red}(\mathit{incompl}) \land \texttt{g} \neq \texttt{h}) \rightarrow \texttt{red}(\texttt{g}) \neq \texttt{red}(\texttt{h})$$

Der Beweis dafür, daß die resultierende Spezifikation tatsächlich den Datentypen **CLIQUE-C** monomorph unter loser Semantik spezifiziert, verläuft völlig analog zum Beweis von Lemma 4.4.

Um zu zeigen, daß **CLIQUE-C** nicht ohne Quantoren spezifizierbar ist, kann man zwischen zwei verschiedene Techniken wählen. Zum einen läßt sich der Beweis von Lemma 4.5 großenteils unverändert übertragen. Man sieht, daß die hinzukommende Funktion `noedge` keine neuen Probleme bringt, da sie entsprechend der obigen Definition einfach einer Ungleichung entspricht, die in einer QF-Spezifikation ohnehin ausgedrückt werden kann. Auch die Verwendung der Reduktionsfunktion `red` anstelle einer Funktion mit booleschem Wertebereich läßt den Beweis nicht deutlich aufwendiger werden — anschaulich gesprochen entspricht der Wert $\texttt{red}(\mathit{incompl})$ ja gerade der Konstante `false` der Sorte `Bool`.

Da die Beweise für beide Aussagen so ähnlich sind, bietet sich als Alternative zu einer Wiederholung des Beweises auch ein Transformationsbeweis an. Man zeigt, wie aus einer monomorphen QF-Spezifikation des Datentyps **CLIQUE-C** eine monomorphe Spezifikation des Datentyps **CLIQUE** gewonnen werden könnte, also ein Beweis durch Reduktion.

Um nachzuweisen, daß **CLIQUE-C** nicht EL-spezifizierbar ist, muß man zu jeder EL-Spezifikation SP Nicht-Standard-Modelle **CLIQUE-C**$_m$ konstruieren, diesmal allerdings so, daß zusätzliche Graphen als *vollständig* charakterisiert werden. Anschließend zeigt man, daß es keinen Σ_{C}-Homomorphismus zwischen **CLIQUE-C** und **CLIQUE-C**$_m$ gibt, so daß nach Definition **CLIQUE-C** nicht initial in $\mathit{Mod}(SP)$ sein kann.

Wir haben also gesehen, daß es einen Datentypen gibt, der zwar PL- und CEL-, aber weder QF- noch EL-spezifizierbar ist. In [Sch98] werden auch Datentypen aus $(CEL \setminus EL) \setminus PL$ und aus $(CEL \setminus EL) \cap QF$ angegeben. Die Eigenschaft eines Datentypen, in $CEL \setminus EL$ zu liegen, ist also weder hinreichend noch notwendig für lose Spezifizierbarkeit mittels PL oder QF. Man könnte nun hoffen, daß zumindest eine Spezifizierbarkeit mittels reiner Gleichungslogik (zusätzlich zur PL-Spezifizierbarkeit) auch die quantorenfreie Spezifizierbarkeit unter loser Semantik sicherstellt, da die reine Gleichungslogik ja deutlich weniger ausdrucksmächtig als die bedingte Gleichungslogik ist. Im folgenden Abschnitt werden wir sehen, daß auch diese Eigenschaft nicht hinreichend für QF-Spezifizierbarkeit ist.

6.5.3 Gleichungslogik

In diesem Abschnitt wollen wir zeigen, daß selbst die Spezifizierbarkeit eines Datentypen sowohl mit Gleichungslogik unter initialer Semantik als auch mit Prädikatenlogik unter loser Semantik nicht ausreicht, um die quantorenfreie Spezifizierbarkeit unter loser Semantik zu garantieren.

Hierzu werden wir den Datentypen **CLIQUE** erneut modifizieren. Unser Vorgehen ist ähnlich dem des letzten Abschnitts — wir werden wiederum mittels einer Reduktionsfunktion alle nicht-vollständigen Graphen auf das selbe Element abbilden. Allerdings haben wir in diesem Fall keine Vorbedingungen zur Verfügung wie in der Spezifikation CLIQUE-C-I, so daß wir alle Vorbedingungen quasi mit Hilfe zusätzlicher Funktionen in Terme „hineinkodieren“ müssen. Insbesondere gilt dies für die Nicht-Existenz von Kanten. Deswegen werden wir eine Funktion `remove` hinzunehmen, die eine Kante aus einem Graphen entfernt. Außerdem wollen wir Ungleichungen zwischen natürlichen Zahlen ausdrücken und führen daher eine zweistellige Funktion `neq` ein, die sicherstellt, daß der Funktionswert stets vom ersten Argument verschieden ist, daß jedoch bei variierendem zweiten Argument alle anderen Werte angenommen werden. Wir definieren also die Signatur

signature Σ_{E}

import Σ_{GRAPH}

sorts `Cliques`

functions

`neq`	: `Nat` × `Nat`	→	`Nat`
`remove`	: `Nat` × `Nat` × `Graph`	→	`Graph`
`red`	: `Graph`	→	`Cliques`

end signature

und die Σ_{E}-Algebra **CLIQUE-E** durch

$$\begin{array}{lcl}
\mathbf{CLIQUE\text{-}E}|_{\Sigma_{\mathrm{GRAPH}}} & = & \mathbf{GRAPH}, \\
\texttt{Cliques}^{\mathbf{CLIQUE\text{-}E}} & = & \mathcal{V} \cup \{\{(0,1)\}\} \\
\texttt{neq}^{\mathbf{CLIQUE\text{-}E}}(x,y) & = & \begin{cases} y & \text{, falls } x \neq y \\ x+1 & \text{, sonst} \end{cases} \\
\texttt{remove}^{\mathbf{CLIQUE\text{-}E}}(x,y,G) & = & G \setminus \{(x,y)\} \\
\texttt{red}^{\mathbf{CLIQUE\text{-}E}}(G) & = & \begin{cases} G & \text{, falls } G \in \mathcal{V} \\ \{(0,1)\} & \text{, sonst} \end{cases}
\end{array}$$

Um die Spezifikation nicht allzu umfangreich und unübersichtlich werden zu lassen, zerlegen wir sie in zwei Teilspezifikationen. Zunächst wollen wir die Sorten `Nat` und `Graph` mitsamt der darauf erklärten Funktionen in einer Spezifikation GRAPH-E-I spezifizieren, danach in einer Spezifikation CLIQUE-E-I die Sorte `Cliques` mit der Funktion `red`.

specification GRAPH-E-I

sorts $\mathtt{Nat}, \mathtt{Graph}$

functions

$$
\begin{array}{lll}
\mathtt{zero} & : & \to \mathtt{Nat} \\
\mathtt{succ} & : \mathtt{Nat} & \to \mathtt{Nat} \\
\mathtt{neq} & : \mathtt{Nat} \times \mathtt{Nat} & \to \mathtt{Nat} \\
\mathtt{empty} & : & \to \mathtt{Graph} \\
\mathtt{connect} & : \mathtt{Nat} \times \mathtt{Nat} \times \mathtt{Graph} & \to \mathtt{Graph} \\
\mathtt{remove} & : \mathtt{Nat} \times \mathtt{Nat} \times \mathtt{Graph} & \to \mathtt{Graph}
\end{array}
$$

variables

$$
\begin{array}{ll}
\mathtt{x}, \mathtt{y}, \mathtt{u}, \mathtt{v} & : \mathtt{Nat} \\
\mathtt{g} & : \mathtt{Graph}
\end{array}
$$

axioms

$$
\begin{array}{ll}
(1) & \mathtt{neq}(\mathtt{zero}, \mathtt{zero}) = \mathtt{succ}(\mathtt{zero}) \\
(2) & \mathtt{neq}(\mathtt{succ}(\mathtt{x}), \mathtt{zero}) = \mathtt{zero} \\
(3) & \mathtt{neq}(\mathtt{zero}, \mathtt{succ}(\mathtt{x})) = \mathtt{succ}(\mathtt{x}) \\
(4) & \mathtt{neq}(\mathtt{succ}(\mathtt{x}), \mathtt{succ}(\mathtt{y})) = \mathtt{succ}(\mathtt{neq}(\mathtt{x}, \mathtt{y})) \\
(5) & \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g})) = \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g}) \\
(6) & \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{connect}(\mathtt{u}, \mathtt{v}, \mathtt{g})) \\
 & \quad = \mathtt{connect}(\mathtt{u}, \mathtt{v}, \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g})) \\
(7) & \mathtt{remove}(\mathtt{x}, \mathtt{y}, \mathtt{empty}) = \mathtt{empty} \\
(8) & \mathtt{remove}(\mathtt{x}, \mathtt{y}, \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g})) = \mathtt{remove}(\mathtt{x}, \mathtt{y}, \mathtt{g}) \\
(9) & \mathtt{remove}(\mathtt{neq}(\mathtt{x}, \mathtt{u}), \mathtt{v}, \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g})) \\
 & \quad = \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{remove}(\mathtt{neq}(\mathtt{x}, \mathtt{u}), \mathtt{v}, \mathtt{g})) \\
(10) & \mathtt{remove}(\mathtt{u}, \mathtt{neq}(\mathtt{y}, \mathtt{v}), \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{g})) \\
 & \quad = \mathtt{connect}(\mathtt{x}, \mathtt{y}, \mathtt{remove}(\mathtt{x}, \mathtt{neq}(\mathtt{y}, \mathtt{v}), \mathtt{g}))
\end{array}
$$

end specification

specification CLIQUE-E-I

import GRAPH-E-I

sorts $\texttt{Cliques}$

functions $\texttt{red} : \texttt{Graph} \to \texttt{Cliques}$

variables $\texttt{x}, \texttt{y}, \texttt{u}, \texttt{v} : \texttt{Nat}$
$\texttt{g} : \texttt{Graph}$

axioms

$$
\begin{aligned}
(11)\ & \texttt{red}(\texttt{remove}(\texttt{x}, \texttt{neq}(\texttt{x}, \texttt{y}), \\
& \qquad\qquad \texttt{connect}(\texttt{x}, \texttt{u}, \texttt{connect}(\texttt{neq}(\texttt{x}, \texttt{y}), \texttt{v}, \texttt{g})))) \\
& \qquad = \texttt{red}(incompl) \\
(12)\ & \texttt{red}(\texttt{remove}(\texttt{x}, \texttt{neq}(\texttt{x}, \texttt{y}), \\
& \qquad\qquad \texttt{connect}(\texttt{u}, \texttt{x}, \texttt{connect}(\texttt{v}, \texttt{neq}(\texttt{x}, \texttt{y}), \texttt{g})))) \\
& \qquad = \texttt{red}(incompl) \\
(13)\ & \texttt{red}(\texttt{remove}(\texttt{x}, \texttt{y}, \texttt{connect}(\texttt{x}, \texttt{neq}(\texttt{y}, \texttt{u}), \\
& \qquad\qquad\qquad \texttt{connect}(\texttt{neq}(\texttt{x}, \texttt{v}), \texttt{y})))) \\
& \qquad = \texttt{red}(incompl) \\
(14)\ & \texttt{red}(\texttt{remove}(\texttt{x}, \texttt{neq}(\texttt{x}, \texttt{y}), \\
& \qquad\qquad \texttt{connect}(\texttt{u}, \texttt{x}, \texttt{connect}(\texttt{y}, \texttt{v}, \texttt{g})))) \\
& \qquad = \texttt{red}(incompl) \\
(15)\ & \texttt{red}(\texttt{remove}(\texttt{x}, \texttt{x}, \texttt{connect}(\texttt{neq}(\texttt{x}, \texttt{u}), \texttt{x}, \texttt{g}))) \\
& \qquad = \texttt{red}(incompl) \\
(16)\ & \texttt{red}(\texttt{remove}(\texttt{x}, \texttt{x}, \texttt{connect}(\texttt{neq}(\texttt{u}, \texttt{x}), \texttt{x}, \texttt{g}))) \\
& \qquad = \texttt{red}(incompl)
\end{aligned}
$$

end specification

Die Gesamtspezifikation ist offensichtlich etwas umfangreicher geworden. Dies liegt daran, daß die verschiedenen Fälle von Disjunktionen alle in getrennte Axiome gepackt werden mußten und insbesondere die Ungleichheit zweier Kanten jeweils mehrere Fälle erzeugte.

Der Beweis dafür, daß CLIQUE-E-I tatsächlich den gewünschten Datentypen unter initialer Semantik definiert, wäre zwar technisch ziemlich aufwendig, aber würde von der Idee her keine Überraschungen in sich bergen, so daß wir ihn an dieser Stelle nicht detailliert führen.

Wir wollen lediglich anmerken, daß durch die Axiome (1)–(4) die Funktion `neq` induktiv definiert wird. Axiome (5)–(10) spezifizieren die Gleichheit von Graphen. Insbesondere läßt sich mittels der Axiome (7)–(10) zu jedem Term die Gleichheit zu einem Term ohne Auftreten des Funktionssymbols `remove` folgern. Axiome (11)–(16) schließlich spezifizieren die Funktion `red`.

Die Axiome (11)–(14) entsprechen den Axiomen (5)–(8) aus der Spezifikation CLIQUE-C-I, wenn dort `x` und `y` verschieden sind. Für `x` = `y` müssen wir noch getrennte Axiome vorsehen, da ansonsten unter Umständen der leere Graph fälschlicherweise als unvollständig spezifiziert würde.

Auf eine Angabe der PL-Spezifikation wollen wir hier verzichten, da ziemlich offensichtlich sein dürfte, wie eine solche aussehen müßte. Die Spezifikation von `red` ist ja, wie wir oben gesehen haben, schon ohne die zusätzlichen Funktionen möglich, und für die zusätzlichen Funktionen kann einfach die EL-Spezifikation übernommen werden.

Um zu zeigen, daß **CLIQUE-C** nicht quantorenfrei spezifizierbar ist, kann man wiederum entweder den Beweis aus Abschnitt 4.3 übertragen oder aber eine Reduktion auf das zuletzt betrachtete Beispiel wählen. Die erste Variante würde einige weitere technische Fälle mit sich bringen, allerdings im Prinzip mit der gleichen Technik funktionieren. In [Sch98] wird ein Transformationsbeweis für eine etwas unterschiedliche Variante geführt.

In diesem Abschnitt haben wir also gesehen (wenn auch nicht ausführlich bewiesen), daß selbst die Spezifizierbarkeit eines Datentyps mit reiner Gleichungslogik unter initialer Semantik und mit Prädikatenlogik unter loser Semantik nicht sicherstellen, daß der Datentyp auch quantorenfrei unter loser Semantik spezifizierbar ist. Da die Existenz von Beispielen, die sowohl mit EL als auch mit QF spezifizierbar sind, trivial ist (man nehme etwa den Datentyp **BOOL** oder einen beliebigen frei erzeugten Datentypen), zeigt das Ergebnis, daß auch die Spezifizierbarkeit mit Gleichungslogik unter initialer Semantik weder eine notwendige noch eine hinreichende Bedingung für quantorenfreien Spezifizierbarkeit ergibt.

Kapitel 7

Zusammenfassung und Ausblick

In der vorliegenden Arbeit haben wir die Rolle von Quantoren bei der monomorphen Spezifikation berechenbarer, termerzeugter Datentypen ohne Verwendung von Hilfsfunktionen untersucht. Im Vordergrund stand dabei die Spezifikation unter loser Semantik mit voller Prädikatenlogik bzw. Prädikatenlogik ohne Quantoren. Das zentrale Ergebnis der Arbeit ist, daß volle Prädikatenlogik (mit Quantoren) echt ausdrucksstärker ist als quantorenfreie, wenn keine Hilfsfunktionen zugelassen sind.

Zwar zeigen Resultate von Bergstra und Tucker (vgl. [BT82], [BT87]), daß unter Verwendung von Hilfsfunktionen jeder berechenbare Datentyp quantorenfrei spezifizierbar ist. Allerdings ist aus verschiedenen Gründen die Verwendung von Hilfsfunktionen nicht immer erwünscht. Zum einen können entsprechende Mechanismen bei Spezifikationswerkzeugen zu technischen Problemen führen. Zum anderen werden Spezifikationen mit Hilfsfunktionen häufig eher zu funktionalen Programmen, was man unter Umständen vermeiden möchte — insbesondere, wenn die Implementierung selbst in einer funktionalen Programmiersprache erfolgt.

Man wird daher abwägen, ob man bei der Spezifikation von Datentypen tatsächlich Hilfsfunktionen verwenden will. Die Ergebnisse dieser Arbeit zeigen, daß unter Verwendung von Quantoren mehr Datentypen spezifiziert werden können als ohne, so daß beim Entwurf von Spezifikationsmethoden die Bereitstellung von Quantoren empfehlenswert ist. Daß es trotzdem berechenbare, termerzeugte Datentypen gibt, die nicht spezifiziert werden können, wurde in [Ber93, Sch97a, Sch97b] nachgewiesen.

Auch in der endlichen Modelltheorie werden Eigenschaften von Spezifikationen relationaler Strukturen untersucht. Die Kriterien sind hier hauptsächlich die Anzahl der Variablen, Anzahl von Quantoren und Anzahl von Formeln. In Arbeiten wie [Imm82, Imm89] wird ein enger Zusammenhang zur Berechnungskomplexität des durch die Axiome definierten Prädikats auf relationalen Strukturen nachgewiesen.

Ein wesentlicher Unterschied besteht jedoch darin, daß bei diesen Spezifikationen keine Signatursymbole für die zu definierende Eigenschaft relationaler Strukturen zur Verfügung stehen und auch keine Quantoren über diese Strukturen. Damit fällt beispielsweise die Möglichkeit induktiver Definitionen weg. Tatsächlich kann man Beispiele angeben, die nicht im Sinne der endlichen Modelltheorie spezifizierbar sind, jedoch als Funktion auf geeigneten Datentypen. Ein solches Beispiel ist etwa der Zusammenhang eines Graphen als Spezialfall einer relationalen Struktur (vgl. [Gai82]). Führt man ein Signatursymbol `connected` ein und erlaubt Quantifikation über Graphen (wie das ja bei uns der Fall ist), so läßt sich eine PL-Spezifikation des Graphzusammenhangs angeben.

Tatsächlich zeigen die Ergebnisse aus Kapitel 6, daß eine einfache Charakterisierung der quantorenfreien Spezifizierbarkeit anhand der Berechnungskomplexität zu spezifizierender Funktionen im Sinne von Blum nicht möglich ist. Im Bereich notwendiger oder hinreichender Bedingungen für quantorenfreie Spezifizierbarkeit sind sicher noch stärkere Ergebnisse zu erreichen als in dieser Arbeit. Das Feld bietet sich somit für weitere Forschung an, wobei die Untersuchung algebraischer Eigenschaften der Datentypen besonders erfolgversprechend scheint.

Ebenfalls offen sind Fragen nach der Existenz von Hierarchien bezüglich der Quantorenanzahl unter zusätzlichen Forderungen wie Beschränktheit der Sortenzahl oder Fixierung einer Signatur. Insbesondere der letzte Fall läßt auf Beziehungen zu anderen Gebieten der mathematischen Logik hoffen.

Anhang A

Polynomdarstellung der Primzahlen

An dieser Stelle wollen wir die in Kapitel 3 erwähnte Spezifikation des Primzahlprädikats angeben. Sie stützt sich auf das nachfolgende Polynom f, dessen nicht-negative Werte gerade die Primzahlen sind. Das Polynom entstammt der Arbeit [JSWW76] und verwendet 26 Variablen, die der Einfachheit halber mit $a, b, c, \ldots, z$ bezeichnet seien.

$$
\begin{aligned}
& f(a, \ldots, z) \\
= \; & (k+2)\Big(1 \; - (wz+h+j-q)^2 \\
& \qquad - ((gk+2g+k+1)(h+j)+h-z)^2 \\
& \qquad - (2n+p+q+z-e)^2 \\
& \qquad - (16(k+1)^3(k+2)(n+1)^2+1-f^2)^2 \\
& \qquad - (e^3(e+2)(a+1)^2+1-o^2)^2 \\
& \qquad - ((a^2-1)y^2+1-x^2)^2 \\
& \qquad - (16r^2y^4(a^2-1)+1-u^2)^2 \\
& \qquad - (n+l+v-y)^2 \\
& \qquad - \Big(\Big((a+u^2(u^2-a))^2-1\Big)(n+4dy)^2+1-(x+cu)^2\Big)^2 \\
& \qquad - ((a^2-1)l^2+1-m^2)^2 \\
& \qquad - (q+y(a-p-1)+s(2ap+2a-p^2-2p-2)-x)^2 \\
& \qquad - (z+pl(a-p)+t(2ap-p^2-1)-pm)^2 \\
& \qquad - (ai+k+1-l-i)^2 \\
& \qquad - (p+l(a-n-1)+b(2an+2a-n^2-2n-2)-m)^2\Big)
\end{aligned}
$$

Multipliziert man das Polynom vollständig aus und sortiert nach positiven bzw. negativen Koeffizienten, so erhält man eine Darstellung als $f = p - q$, wobei die Koeffizienten der Polynome p, q natürlichzahlig sind. Bezeichnen

t_p, t_q die nach Lemma 3.5 existenten Terme zur Darstellung von p, q, so ist eine vollständige QF-Spezifikation des Primzahlprädikats gegeben durch:

specification PRIM

import NAT-MULT, $\mathtt{Bool}$

functions $\mathtt{prim} : \mathtt{Nat} \to \mathtt{Bool}$

variables $\mathtt{a}, \ldots, \mathtt{z}, \mathtt{z}_1 : \mathtt{Nat}$

axioms

$$\mathtt{prim}(\mathtt{zero}) = \mathtt{false}$$
$$\mathtt{prim}(\mathtt{succ}(\mathtt{zero})) = \mathtt{false}$$
$$\mathtt{prim}(\mathtt{mult}(\mathtt{succ}(\mathtt{succ}(\mathtt{x})), \mathtt{succ}(\mathtt{succ}(\mathtt{y})))) = \mathtt{false}$$
$$\mathtt{add}(\mathtt{z}_1, t_q(\mathtt{a}, \ldots, \mathtt{z})) = t_p(\mathtt{a}, \ldots, \mathtt{z}) \to \mathtt{prim}(\mathtt{z}_1) = \mathtt{true}$$

end specification

Literaturverzeichnis

[AT82] P.R.J. Asveld and J.V. Tucker. Complexity theory and the operational structure of algebraic programming systems. *Acta Informatica*, 17:451–476, 1982.

[BBB+85] F.L. Bauer, R. Berghammer, M. Broy, W. Dosch, R. Gnatz, F. Geiselbrechtinger, E. Hangel, W. Hesse, B. Krieg-Brückner, A. Laut, T.A. Matzner, B. Möller, F. Nickel, H. Partsch, P. Pepper, K. Samelson, M. Wirsing, and H. Wössner. *The Munich Project CIP. Volume I: The Wide Spectrum Language CIP-L*, volume 183 of *LNCS*. Springer, Berlin, 1985.

[Ber93] R. Berghammer. On the characterization of the integers: the hidden function problem revisited. *Acta Cybernetica*, 11(1-2):85–96, 1993.

[Bör89] E. Börger. *Computability, Complexity, Logic*, volume 128 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Publishers B. V., 1989.

[BT82] J.A. Bergstra and J.V. Tucker. The completeness of the algebraic specification methods for computable data types. *Information and Control*, 54:186–200, 1982.

[BT83] J.A. Bergstra and J.V. Tucker. Initial and final algebra semantics for data type specifications: Two characterization theorems. *SIAM J. Comput.*, 12:366–387, 1983.

[BT87] J.A. Bergstra and J.V. Tucker. Algebraic specifications of computable and semicomputable data types. *Theoret. Comput. Sci.*, 50:137–181, 1987.

[Cla89] I. Claßen. Revised ACT ONE: categorical constructions for an algebraic specification language. In *Proc. Workshop on Categorical Methods in Computer Science with Aspects from Topology,* *Berlin*, volume 393 of *LNCS*, pages 124–141. Springer, Berlin, 1989.

[Cut80] N. J. Cutland. *Computability. An introduction to recursive function theory.* Cambridge University Press, 1980.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer, Berlin, 1995.

[EGL89] H.-D. Ehrich, M. Gogolla, and U.W. Lipeck. *Algebraische Spezifikation abstrakter Datentypen.* Verlag B.G. Teubner, 1989.

[EM81] H. Ehrig and B. Mahr. Complexity of algebraic implementations for abstract data types. *Journal of Computer and System Sciences*, 23:223–253, 1981.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1, Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science.* Springer, Berlin, 1985.

[Gai82] H. Gaifman. On local and non-local properties. In J. Stern, editor, *Proc. of the Herbrand Symposium, Logic Colloquium'81*, pages 105–135. North-Holland, 1982.

[GS95] D. Gries and F.B. Schneider. Avoiding the undefined by underspecification. In *Computer Science Today: Recent Trends and Developments*, volume 1000 of *LNCS*, pages 366–373. Springer, Berlin, 1995.

[GTW78] J.A. Goguen, J.W. Thatcher, and E.G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology IV: Data Structuring*, pages 80–144, 1978.

[HMRS90] M. Heisel, W. Menzel, W. Reif, and W. Stephan. *Der Karlsruhe Interactive Verifier (KIV). Eine Übersicht*, pages 172–193. Hüthig Verlag, 1990.

[Imm82] N. Immerman. Upper and lower bounds for first order expressibility. *Journal of Computer and System Sciences*, 25:76–98, 1982.

[Imm89] N. Immerman. Descriptive and computational complexity. In J. Hartmanis, editor, *Proceedings of Symposia in Applied Math., Vol.38*, pages 75–91. American Mathematical Society, 1989.

[JSWW76] J. P. Jones, D. Sato, H. Wada, and D. Wiens. Diophantine representation of the set of prime numbers. *The American Mathematical Monthly*, 83(6):449–464, 1976.

[Kle81] S.C. Kleene. Origins of recursive function theory. *Annals of the History of Computing*, 3(1), 1981.

[Kum92] Martin Kummer. Komplexität. Universität Karlsruhe, Fakultät für Informatik, 1992. Skriptum zur gleichnamigen Vorlesung.

[LEW96] J. Loeckx, H.-D. Ehrich, and M. Wolf. *Specification of abstract data types*. Verlag Wiley-Teubner, 1996.

[Mal61] A.I. Malćev. Constructive algebras. In *Russian Math. Surveys*, volume 16, pages 77–129, 1961.

[Man77] Y. Manin. *A course in Mathematical Logic*. Springer-Verlag, Berlin, 1977.

[Mat70] Y. Matiyasevich. Enumerable sets are diophantine. *Soviet Mathematics. Doklady*, 11(2):354–358, 1970.

[Mat93] Y. Matiyasevich. *Hilbert's Tenth Problem*. Foundation of Computing Series. MIT Press, 1993.

[MG85] J. Meseguer and J.A. Goguen. Initiality, induction, and computability. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge University Press, 1985.

[Odi89] P. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Publishers B. V., 1989.

[Pau78] W. J. Paul. *Komplexitätstheorie*. Teubner, 1978.

[Rab60] M.O. Rabin. Computable algebra, general theory and the theory of computable fields. In *Trans. Amer. Math. Soc.*, volume 98, pages 341–360, 1960.

[Rei91] W. Reif. *Korrektheit von Spezifikationen und generischen Moduln*. PhD thesis, Universität Karlsruhe, Fakultät für Informatik, 1991.

[Rei92] W. Reif. Correctness of full first-order specifications. In *4th Conference on Software Engineering and Knowledge Engineering*. Capri, Italy, IEEE Press, 1992.

[Rei95] W. Reif. The KIV-Approach to Software Verification. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages, and Tools for the Construction of Correct Software*, volume 1009 of *LNCS*. Springer, Berlin, 1995.

[Rob74] A. Robinson. *Introduction to model theory and to the metamathematics of algebra*. Studies in Logic and the Foundations of Mathematics. North Holland Publishers, 1974.

[Rog67] H. Rogers. *Theory of Recursive Functions and Effective Computability*. Series in Higher Mathematics. McGraw-Hill Book Company, 1967.

[Sch97a] A. Schönegge. An answer to the hidden function question for algebraic specification methods (abstract), 4th Workshop on Logic, Language, Information and Computation, Fortaleza, Brazil. *Logic Journal of the Interest Group in Pure and Applied Logic, Oxford University Press*, 5(6), 1997.

[Sch97b] A. Schönegge. The hidden function question revisited. In *Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology (AMAST '97), Sydney, Australia*, volume 1349 of *LNCS*, pages 451–464. Springer, Berlin, 1997.

[Sch98] A. Schönegge. *Spezifizierbarkeit berechenbarer Datentypen*. PhD thesis, Universität Karlsruhe, Fakultät für Informatik, 1998.

[Sho67] J. Shoenfield. *Mathematical Logic*. Addison-Wesley, 1967.

[SW83] D.T. Sannella and M. Wirsing. A kernel language for algebraic specification and implementation. In *Proc. 1983 International Conference on Foundations of Computation Theory, Borgholm*, volume 158 of *LNCS*, pages 413–427. Springer, Berlin, 1983.

[TWW82] J.W. Thatcher, E.G. Wagner, and J.B. Wright. Data type specification: Parameterization and the power of specification techniques. In *ACM Transactions on Programming Languages and Systems*, volume 4, pages 711–732, 1982.

[Wir90] M. Wirsing. *Algebraic Specification*, volume B of *Handbook of Theoretical Computer Science*, chapter 13, pages 675–788. Elsevier Science Publishers B. V., 1990.