Gottfried Wilhelm Leibniz Universität Hannover
Institut für Theoretische Informatik

Algebraische Charakterisierungen von Komplexitätsklassen

Bachelorarbeit

Lenard Ehrmuth
Matrikelnr. 10030868

Hannover, den 2. September 2022

Erstprüfer: Prof. Dr. rer. nat. Heribert Vollmer
Zweitprüfer: Dr. rer. nat. habil. Arne Meier
Betreuer: Prof. Dr. rer. nat. Heribert Vollmer

Einleitung

Turingmaschinen sind ein mächtiges Werkzeug. Nicht nur, um Berechnungen durchzuführen, sondern auch auf der theoretischen Seite. Die intuitive Definition von Komplexitätsklassen wie **P** oder **NP** verwendet solche Maschinen und nach der Church-Turing-These lassen sich alle intuitiv berechenbaren Funktionen von ihnen berechnen. Doch der Formalismus hinter diesen Modellen wirkt meist eher kompliziert mit langen Beweisen aus größtenteils undefinierten Algorithmen.
Die Frage, um die es in dieser Arbeit geht, ist, ob die berechneten Funktionen selbst inhärente Eigenschaften besitzen, durch die sie sich auch ohne ein zugrundeliegendes Maschinenmodell klassifizieren lassen. Wenn man sich mit Klassifizierungen von Funktionen auf den natürlichen Zahlen beschäftigt, stößt man früher oder später auf den polnischen Mathematiker Andrzej Grzegorczyk und sein Paper „*Some classes of recursive functions*“ [2] aus dem Jahr 1953. Darin beschreibt er mit rein mathematischen Mitteln eine Hierarchie von Funktionsklassen. Die Beweisprizipien beruhen dabei hauptsächlich auf vollständiger Induktion.
Der Zusammenhang zu Turingmaschinen wurde darauf hin verdeutlicht von Robert W. Ritchie in „Classes of predictably computable Functions“ [4] (1963) und Alan Cobham mit „The Intrinsic Computational Difficulty of Functions“ [1] (1965). Gemeinsam bildeten sie einen Grundstein der heutigen Komplexitätstheorie. Diese Arbeit ist eine Zusammenfassung dieser drei historischen Werke aus einer modernen Perspektive. Wir werden zeigen, wie man Klassen von Funktionen algebraisch Charakterisiert, wie daraus die sogenannte Grzegorczyk Hierarchie der primitiv-rekursiven Funktionen entstanden ist und dass ihre Klassen größtenteils mit bestimmten Komplexitätsklassen auf Turingmaschinen übereinstimmen.
Zuletzt werden wir noch darauf eingehen, wieso dieser Übereinstimmung eine gewisse Grenze gesetzt ist und warum Cobham aus diesem Grund die Klasse der Polynomialzeitfunktionen definiert hat, die in der weiteren Entwicklung der Komplexitätstheorie bis heute eine sehr wichtige Rolle spielt.

Inhaltsverzeichnis

1 Algebra

Dieses erste Kapitel erklärt die mathematischen Grundlagen, die wir brauchen, um unendliche Mengen von allgemeinen Funktionen mit relativ einfachen Begriffen zu charakterisieren. Als Grundlage dient das Kapitel 1.3 aus dem Buch „Theoretische Informatik: Eine kompakte Einführung“ von Klaus W. Wagner.

1.1 Hüllenoperatoren

Ein hilfreiches mathematisches Werkzeug für die Informatik ist die algebraische Erzeugung. Sie ist in der Lage die komplexe Struktur einer unendlichen Menge auf endlich viele Elemente zurückzuführen. Ein Beispiel dafür ist der Begriff der linearen Hülle:

$$\mathbb{R}^3 = \langle \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \rangle$$

Um dieses Konzept zu verallgemeinern, definieren wir als erstes den Begriff des Hüllenoperators.

Definition 1 ([6, S. 12]). Eine Funktion $\Gamma : \mathcal{P}(\mathbf{D}) \to \mathcal{P}(\mathbf{D})$ heißt Hüllenoperator (oder Abschlussoperator) über $\mathbf{D}$, wenn sie die folgenden Eigenschaften besitzt:

1. Einbettung: Für alle $A \in \mathcal{P}(\mathbf{D})$ gilt $A \subseteq \Gamma(A)$
2. Monotonie: Für alle $A, B \in \mathcal{P}(\mathbf{D})$ gilt $A \subseteq B \Rightarrow \Gamma(A) \subseteq \Gamma(B)$
3. Abgeschlossenheit: Für alle $A \in \mathcal{P}(\mathbf{D})$ gilt $\Gamma(\Gamma(A)) = \Gamma(A)$

Beispiel 1.1. Die lineare Hülle ist ein Hüllenoperator.

Beispiel 1.2. Die transitive Hülle einer Relation, also die kleinste Menge, die R als Teilmenge hat und transitiv ist, ist ein Hüllenoperator.

Für die nächste Definition habe ich die Notation etwas angepasst und die Einschränkung der Definitionsbereiche entfernt.

Definition 2 ([6, S. 12]). Aus einer Menge von Funktionen $\mathbf{O}$, die in diesem Kontext als Operation bezeichnet wird, kann ein Hüllenoperator konstruiert werden. Die Funktionen können verschiedene Anzahlen von Parametern und sogar verschiedene Definitionsbereiche haben. Für alle $f \in \mathbf{O}$ gibt es $n_f \in \mathbb{N}$ und $D_1^f, ..., D_{n_f}^f$, sodass:

$$f : D_1^f \times ... \times D_{n_f}^f \to \mathbf{D}$$

Daraus ergibt sich die folgende induktive Definition:

$$\begin{aligned}
\Gamma_{\mathbf{O}}^0(M) &= M \\
\Gamma_{\mathbf{O}}^{k+1}(M) &= \Gamma_{\mathbf{O}}^k(M) \cup \bigcup_{f \in \mathbf{O}} f\left(\Gamma_{\mathbf{O}}^k(M)^{n_f} \cap (D_1^f \times ... \times D_{n_f}^f)\right) \\
\Gamma_{\mathbf{O}}(M) &= \bigcup_{k=0}^{\infty} \Gamma_{\mathbf{O}}^k(M)
\end{aligned}$$

$f(A)$ für eine Menge A beschreibt das Bild von A unter f. In jedem Schritt werden also die Funktionswerte aller $f \in \mathbf{O}$ gebildet, die mit Werten aus dem vorigen Schritt als Argumente entstehen können. Eine Menge heißt abgeschlossen unter der Operation $\mathbf{O}$, wenn alle möglichen Funktionswerte bereits in der Menge enthalten sind.

Satz 3. *Für alle* $\mathbf{D}$ *und für alle* $\mathbf{O}$ *ist* $\Gamma_{\mathbf{O}}$ *ein Hüllenoperator auf* $\mathbf{D}$.

Beweis. 1. $A = \Gamma_{\mathbf{O}}^0(A) \subseteq \bigcup_{k=0}^{\infty} \Gamma_{\mathbf{O}}^k(A) = \Gamma_{\mathbf{O}}(A)$

2. Aus $A \subseteq B$ folgt nach **Induktion** über k:

$$\Gamma_{\mathbf{O}}^0(A) = A \subseteq B = \Gamma_{\mathbf{O}}^0(B)$$

Da das Bild von Funktionen sich monoton verhält, gilt:

$$\Gamma_{\mathbf{O}}^k(A) \subseteq \Gamma_{\mathbf{O}}^k(B) \quad \Rightarrow \quad \Gamma_{\mathbf{O}}^{k+1}(A) \subseteq \Gamma_{\mathbf{O}}^{k+1}(B)$$

Insgesamt gilt also für alle k:

$$\Gamma_{\mathbf{O}}^k(A) \subseteq \Gamma_{\mathbf{O}}^k(B) \subseteq \Gamma_{\mathbf{O}}(B) \quad \Rightarrow \quad \Gamma_{\mathbf{O}}(A) \subseteq \Gamma_{\mathbf{O}}(B)$$

3. Für alle $f \in \mathbf{O}$ gilt: Für alle $x \in \Gamma_{\mathbf{O}}(A)$ gibt es ein k, sodass $x \in \Gamma_{\mathbf{O}}^k(A)$. Dann gibt es auch für alle $(x_1, ..., x_{n_f}) \in \Gamma_{\mathbf{O}}(A)^{n_f} \cap (D_1^f \times ... \times D_{n_f}^f)$ ein k, sodass $x_1, ..., x_{n_f} \in \Gamma_{\mathbf{O}}^k(A)$. Daraus folgt:

$$f(x_1, ..., x_{n_f}) \in \Gamma_{\mathbf{O}}^{k+1}(A) \subseteq \Gamma_{\mathbf{O}}(A)$$

Folglich liegen alle von f erzeugten Elemente für $\Gamma_{\mathbf{O}}(\Gamma_{\mathbf{O}}(A))$ bereits in $\Gamma_{\mathbf{O}}(A)$. Daraus folgt $\Gamma_{\mathbf{O}}(\Gamma_{\mathbf{O}}(A)) \subseteq \Gamma_{\mathbf{O}}(A)$ und mit 1.: $\Gamma_{\mathbf{O}}(\Gamma_{\mathbf{O}}(A)) = \Gamma_{\mathbf{O}}(A)$ □

Definition 4. Des Weiteren definieren wir eine etwas praktischere Schreibweise für endlich viele $x_1, \dots, x_n \in \mathbf{D}$ und die Operationen $O_1, \dots, O_m$:

$$[x_1, \dots, x_n \,;\, O_1, \dots, O_m] = \Gamma_{O_1 \cup \dots \cup O_m}(\{x_1, \dots, x_n\})$$

Wenn die Funktionen von nur einer Operation direkt angegeben werden, dann sei:

$$[x_1, \dots, x_n \,;\, f_1, \dots, f_m] = \Gamma_{\{f_1, \dots, f_m\}}(\{x_1, \dots, x_n\})$$

Eine auf diese Weise definierte Hülle ist immer die kleinste Teilmenge von $\mathbf{D}$, die die Elemente $x_1, \dots, x_n$ enthält und abgeschlossen ist unter den Operationen $O_1, \dots, O_m$. Denn die Hülle enthält genau die Elemente, die sich mit den Operationen $O_1, \dots, O_m$ aus $x_1, \dots, x_n$ bilden lassen.

Beispiel 4.1. Die natürlichen Zahlen sind nach ihrer Definition algebraisch erzeugt.

$$\mathbb{N} = [0 \,;\, x + 1]$$

Beispiel 4.2. Fügt man die Funktion $x - 1$ hinzu, erhält man die ganzen Zahlen.

$$\mathbb{Z} = [0 \,;\, x + 1, x - 1]$$

Als Obermenge für diese beiden Beispiele eignet sich $\mathbb{R}$ oder sogar $\mathbb{C}$.

1.2 Induktion

Definition 5. Die Ordnung von $x \in \Gamma_{\mathbf{O}}(M)$ ist das kleinste $k \in \mathbb{N}$, sodass $x \in \Gamma^k_{\mathbf{O}}(M)$. Die Ordnung von x entspricht der kleinsten Anzahl von Schritten aus den Operationen, die man benötigt, um x aus der Initialmenge zu erzeugen. Es sei angemerkt, dass die Ordnung abhängig von der gewählten Charakterisierung der Menge ist.

Beispiel 5.1. Die Menge $\mathbb{Z}$ kann auf mehrere Arten charakterisiert werden:

$$\mathbb{Z} = [0 \,;\, x + 1, x - 1] = [0, 1 \,;\, x + 1, x - 1]$$

Die 1 hat in $[0 \,;\, x + 1, x - 1]$ Ordnung 1 und in $[0, 1 \,;\, x + 1, x - 1]$ Ordnung 0. Dadurch ändern sich auch die Ordnungen aller positiven Zahlen um 1.

Beispiel 5.2. Auch die Operationen haben Einfluss auf die Ordnung:

$$\mathbb{Z} = [0\,;\, x+1, x-1] = [0\,;\, x+1, x-1, x+y]$$

In beiden Fällen hat die Zahl 2 die Ordnung 2, doch in $[0\,;\, x+y, x-1]$ kann 4 aus $2+2$ gebildet werden. Dadurch erhält 4 die Ordnung 3, aber in $[0\,;\, x+1, x-1]$ hat 4 die Ordnung 4.

Der Begriff einer Ordnung ist hilfreich, um Aussagen auf der gesamten Menge durch vollständige **Induktion** zu beweisen [vgl. 6, S. 17].

Beispiel 5.3. Für alle $x \in [0\,;\, x+1, x-1] = \mathbb{Z}$ der Ordnung k gilt $|x| \leq k$.

Beweis. **Induktion** über k:

Die Elemente mit Ordnung 0 sind $\{0\}$ und $|0| \leq 0$.

Die Elemente mit Ordnung k sind:

$$\begin{aligned}&\Gamma^k_{\{x+1,x-1\}}(\{0\}) \setminus \Gamma^{k-1}_{\{x+1,x-1\}}(\{0\}) \\ &\qquad = \{x+1 \mid x \in \Gamma^{k-1}_{\{x+1,x-1\}}(\{0\})\} \cup \{x-1 \mid x \in \Gamma^{k-1}_{\{x+1,x-1\}}(\{0\})\}\end{aligned}$$

Die Elemente, deren Ordnung kleiner als k ist sind genau $\Gamma^{k-1}_{\{x+1,x-1\}}(\{0\})$. Wir nehmen nun an, dass für alle diese Elemente $|x| \leq k-1$ gilt. Daraus folgt:

$$\begin{aligned}|x+1| &\leq |x| + |1| \quad &\leq k-1+1 = k \\ |x-1| &\leq |x| + |-1| &\leq k-1+1 = k\end{aligned}$$

Also gilt auch für alle $y \in \Gamma^k_{\{x+1,x-1\}}(\{0\})$: $|y| \leq k$.

Nach diesem Induktionsschritt gilt für x jeder Ordnung k: $|x| \leq k$. □

Man kann also Aussagen über die Menge $[x_1, \ldots, x_n\,;\, O_1, \ldots, O_m]$ beweisen, indem man sie für $x_1, \ldots, x_n$ beweist und dann zeigt, dass alle Operationen die Aussage erhalten.

1.3 Funktionen

Man kann das Konzept der algebraischen Erzeugung natürlich nicht nur auf Zahlen oder Vektoren anwenden, sondern für jede beliebige Obermenge. Interessant für die Komplexitätstheorie sind dabei Mengen von Funktionen.

Definition 6. Eine Funktionsklasse über einer Menge **D** ist eine Teilmenge

$$\mathcal{X} \subseteq \{f \mid f : \mathbf{D}^n \to \mathbf{D} \wedge n \in \mathbb{N} \setminus \{0\}\},$$

also eine Menge von Funktionen mit beliebig vielen Parametern auf **D**.

Beispiel 6.1. Der Polynomring über $\mathbb{Z}$ ist $\mathbb{Z}[X] = [-1, X\,;\, x + y, x * y]$. Die Menge aller Polynomfunktionen über $\mathbb{Z}$ lässt sich ebenfalls auf diese Weise konstruieren:

$$\mathcal{X} = [f(x) = -1, g(x) = x\,;\, f + g, f * g]$$

Die Operation umfasst hier allerdings die Addition und Multiplikation von Funktionen.

Auf Funktionen lassen sich aber auch komplexere Operationen definieren, die mit Zahlen nicht funktionieren. Ein solches Beispiel ist die Komposition.

Definition 7. Eine Klasse $\mathcal{X}$ von Funktionen heißt abgeschlossen unter **Komposition**, wenn für alle einstelligen $f(x), g(x) \in \mathcal{X}$ gilt:

$$h(x) = (f \circ g)(x) = f(g(x)) \in \mathcal{X}$$

Diese Definition beschreibt indirekt die Operation, unter der $\mathcal{X}$ abgeschlossen ist:

$$O = \{\circ : (f(x), g(x)) \mapsto f(g(x))\}$$

Die Operation wird später bezeichnet mit $\circ$.

Beispiel 7.1. Diese Polynomfunktionen über $\mathbb{Z}$ sind abgeschlossen unter Komposition.

Beweis. Sei $\mathcal{X}$ die Menge der Polynomfunktionen über $\mathbb{Z}$. Dann gilt für alle $p(x) \in \mathcal{X}$ und $q(x) = a_0 * x^0 + ... + a_n * x^n \in \mathcal{X}$:

$$\begin{aligned} p(x)^2 = p(x) * p(x) &\in \mathcal{X} \\ \Rightarrow p(x)^1, ..., p(x)^n &\in \mathcal{X} \\ \Rightarrow a_1 * p(x)^1, ..., a_n * p(x)^n &\in \mathcal{X} \\ \Rightarrow a_0 * p(x)^0 + a_1 * p(x)^1 + ... + a_n * p(x)^n = q(p(x)) &\in \mathcal{X} \end{aligned}$$

□

Das Werkzeug der algebraischen Erzeugung können wir also zur algebraischen Charakterisierung von verschiedenen Funktionsklassen verwenden. Die Klassen erhalten dadurch eine Art von Struktur, die es deutlich einfacher macht, damit zu arbeiten. Wenn man eine Klasse algebraisch charakterisiert hat, lässt sie sich leicht mit ähnlich charakterisierten Klassen vergleichen, ohne dass man dabei Aussagen über allgemeine Funktionen oder sogar allgemeine Turingmaschinen beweisen muss.

2 Die Grzegorczyk Hierarchie

Andrzej Grzegorczyk war ein polnischer Mathematiker und zu der Zeit, als er „*Some classes of recursive functions*“[2] veröffentlichte, hat die Informatik gerade begonnen, zu einer eigenständigen Wissenschaft zu werden. Das zweite Kapitel ist eine Zusammenfassung dieses Werks. Alle alleinstehenden Seitenzahlen beziehen sich darauf. Stellenweise habe ich auch ein paar Änderungen Vorgenommen. Betreffen diese lediglich die Notation, wie das Umbenennen einer Funktion, dann werden sie nicht explizit erwähnt.

2.1 Klassen von Funktionen über $\mathbb{N}$

2.1.1 Basisfunktionen

Im Allgemeinen betrachtet Grzegorczyk Funktionen von beliebig vielen Parametern auf den natürlichen Zahlen $\mathbb{N}$ mit $0 \in \mathbb{N}$. Das heißt, jede Funktion f ist von der Form:

$$f : \mathbb{N}^k \to \mathbb{N},\ (x_1, \dots, x_k) \mapsto f(x_1, \dots, x_k)$$

Außerdem sind alle betrachteten Funktionen total, also definiert für jede Kombination natürlicher Argumente einschließlich 0.
In der folgenden Betrachtung der elementaren Funktionen werden einige Funktionen benötigt, die häufig auftauchen. Darunter fallen:

$$S(x) = x + 1, \text{ die Nachfolgerfunktion}$$

$$x + y, \text{ die Addition auf den natürlichen Zahlen}$$

$$x \dot{-} y = \begin{cases} x - y, & \text{für } x \geq y \\ 0, & \text{für } x < y \end{cases}, \text{ die Subtraktion auf den natürlichen Zahlen}$$

$$x * y, \text{ die Multiplikation auf den natürlichen Zahlen}$$

$$x^y = \prod_{1}^{y} x, \text{ die Potenzfunktion}$$

$$x! = \prod_{i=1}^{x} i, \text{ die Fakultätsfunktion}$$

2.1.2 Operationen

Auf der Menge der totalen Funktionen auf $\mathbb{N}$ definieren wir nun einige Operationen.

Definition 8 (S. 5). Die Funktionsklasse $\mathcal{X}$ heißt abgeschlossen unter den **Operationen der Substitution**, wenn sie unter den folgenden drei Operationen abgeschlossen ist:

1. **Substitution (Superposition) von Funktionen** Wenn $\mathcal{X}$ die Funktionen $f(x_1, \dots, x_{k-1}, x_k, x_{k+1}, \dots, x_n)$ und $g(y_1, \dots, y_m)$ enthält, enthält $\mathcal{X}$ auch die Substitution der k-ten Variable von f durch g:

$$h(x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_n, y_1, \dots, y_m) \\ = f(x_1, \dots, x_{k-1}, g(y_1, \dots, y_m), x_{k+1}, \dots, x_n)$$

2. **Identifikation von Variablen** Wenn $\mathcal{X}$ die Funktion $f(x_1, \dots, x_i, \dots, x_j, \dots, x_n)$ enthält, enthält $\mathcal{X}$ auch die Funktion

$$h(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_{j-1}, x_{j+1}, \dots, x_n, y) \\ = f(x_1, \dots, x_{i-1}, y, x_{i+1}, \dots, x_{j-1}, y, x_{j+1}, \dots, x_n)$$

   die durch Identifikation der Variablen x_i und x_j mit y entsteht.
   Dabei ist $y \neq x_k$ für alle $k \in \{1, \dots, n\}$.

3. **Substitution von Konstanten** Wenn $\mathcal{X}$ die Funktion $f(x_1, \dots, x_{k-1}, x_k, x_{k+1}, \dots, x_n)$ enthält, enthält $\mathcal{X}$ auch die Substitution der k-ten Variable von f durch 0:

$$h(x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_n) = f(x_1, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n)$$

Die Operationen der Substitution werden im folgenden nur noch vereinigt unter der Bezeichnung **Substitution** oder *Sub* betrachtet.

Beispiel 8.1. Aus den Funktionen $f(x, y) = x + y$ und $S(z) = z + 1$ entsteht durch Substitution von y durch $S(x)$ die Funktion

$$g(x, z) = f(x, S(z)) = x + (z + 1).$$

Beispiel 8.2. Aus der Funktion $g(x, z) = x + (z + 1)$ entsteht durch Identifikation der Variablen x und z die Funktion

$$h(w) = g(w, w) = w + (w + 1) = 2 * w + 1.$$

Beispiel 8.3. Aus der Funktion $u(x, y) = x \dot{-} y$ entsteht durch Substitution von y durch $S(z)$ die Funktion $v(x, z) = u(x, S(z)) = x \dot{-} (z+1)$. Daraus entsteht durch Substitution von z durch 0 die Funktion

$$P(x) = v(x, 0) = x \dot{-} (0 + 1) = x \dot{-} 1,$$

die Vorgängerfunktion auf den natürlichen Zahlen (mit $P(0) = 0$).

Beispiel 8.4 (S. 6). Wenn eine Klasse $\mathcal{X}$ $x + 1$ und $x + y$ enthält und abgeschlossen unter Substitution ist, dann enthält $\mathcal{X}$ auch jede lineare Funktion

$$f(x_1, \dots, x_n) = k_0 + k_1 * x_1, \dots, k_n * x_n \quad \text{für } k_1, \dots, k_n > 0.$$

Beweis. Nach Substitution von 0 ist die Identität $I(y) = 0 + y$ in $\mathcal{X}$ enthalten.
Induktion über k:

Für $k = 1$ ist $f(x_1) = k * x_1 = I(x_1) \in \mathcal{X}$.

Wenn $f(x_1) = k * x_1$ in $\mathcal{X}$ enthalten ist, dann auch $f(x_1) + y$.
(Durch Substitution aus $x + y$ und $f(x_1)$)
Dann ist auch $f'(x_1) = f(x_1) + x_1 = (k + 1) * x_1$ in $\mathcal{X}$ enthalten.

Daraus folgt für alle $k \in \mathbb{N}$: $f(x_1) = k * x_1 \in \mathcal{X}$.
Nach Substitution gilt auch für alle $k_0, k_1 \in \mathbb{N}$:

$$\begin{gathered} x + f(x_1) \in \mathcal{X} \\ \Rightarrow f''(x_0, x_1) = f(x_0) + f(x_1) = k_0 * x_0 + k_1 * x_1 \in \mathcal{X} \\ \Rightarrow f''(x + 1, x_1) \in \mathcal{X} \\ \Rightarrow f_1(x_1) = f''(0 + 1, x_1) = k_0 * (0 + 1) + k_1 * x_1 = k_0 + k_1 * x_1 \in \mathcal{X} \end{gathered}$$

Induktion über n:

Für $n = 1$ ist $f_n(x_1) = f_1(x_1) \in \mathcal{X}$.

$f_n(x_1, \dots, x_n) = k_0 + k_1 * x_1, \dots, k_n * x_n \in \mathcal{X}$

$$\begin{aligned} &\Rightarrow f_n(x_1, \dots, x_n) + y \in \mathcal{X} \\ &\Rightarrow f_{n+1}(x_1, \dots, x_{n+1}) = f_n(x_1, \dots, x_n) + f(x_{n+1}) \in \mathcal{X} \\ &\Leftrightarrow f_{n+1}(x_1, \dots, x_{n+1}) = k_0 + k_1 * x_1, \dots, k_{n+1} * x_{n+1} \in \mathcal{X} \end{aligned}$$

Folglich gilt für alle $n \geq 1, k_1, \dots, k_n > 0$:

$$f(x_1, \dots, x_n) = k_0 + k_1 * x_1, \dots, k_n * x_n \in \mathcal{X} \qquad \square$$

Die Klasse $\mathcal{A} = \{f(x_1, ..., x_n) = k_0 + k_1 * x_1, ..., k_n * x_n\}$ der Linearen Funktionen lässt sich also charakterisieren durch die algebraische Hülle

$$\mathcal{A} = [x + 1, x + y\,;\, Sub]$$

In der weiteren Notation dieser Arbeit stehen Funktionsparameter mit Vektorpfeilen für eine beliebige Anzahl von Parametern. Das heißt, $f(\vec{x}, y)$ beschreibt für alle $k \in \mathbb{N}$ eine Funktion von $k + 1$ Parametern $f(x_1, ..., x_k, y)$. Darunter fällt auch $f(y)$ für $k = 0$.

Definition 9 (S. 8). Eine Funktionsklasse heißt abgeschlossen unter der **Operation der beschränkten Addition**, wenn sie für jede Funktion $f(x_1, ..., x_n, i)$ auch die Funktion

$$g(\vec{u}, y) = \sum_{i \leq y} f(\vec{u}, i) = f(\vec{u}, 0) + ... + f(\vec{u}, y)$$

enthält. Im folgenden wird diese Operation abgekürzt durch $lim. \sum$.

Definition 10 (S. 8). Eine Funktionsklasse heißt abgeschlossen unter der **Operation der beschränkten Multiplikation**, wenn sie für jede Funktion $f(x_1, ..., x_n, i)$ auch die Funktion

$$g(\vec{u}, y) = \prod_{i \leq y} f(\vec{u}, i) = f(\vec{u}, 0) * ... * f(\vec{u}, y)$$

enthält. Im folgenden wird diese Operation abgekürzt durch $lim. \prod$.

2.1.3 Relationen

Neben Funktionen gibt es in der Mathematik noch ein weiteres Konzept, welches überall Anwendungen findet und das sind die Relationen. Mit Hilfe von n-stelligen Funktionen lassen sich auch n-stellige Relationen beschreiben. Dazu verwenden wir die folgende Definition.

Definition 11 (S. 6). Eine n-stellige Relation $R(x_1, ..., x_n)$ ist eine Relation der Funktionsklasse $\mathcal{X}$ genau dann, wenn es eine Funktion $f(x_1, ..., x_n) \in \mathcal{X}$ gibt, sodass für alle $x_1, ..., x_n \in \mathbb{N}$ gilt

$$R(x_1, ..., x_n) \equiv f(x_1, ..., x_n) = 0.$$

Das Äquivalenzzeichen $\equiv$ bezeichnet hier die Äquivalenz prädikatenlogischer Formeln.

Beispiel 11.1 (S. 8). Die Ordnungsrelationen auf $\mathbb{N}$ lassen sich wie folgt ausdrücken:

$$\begin{aligned}
x \leq y &\equiv x \dot{-} y = 0 \\
x < y \equiv x + 1 \leq y &\equiv (x+1) \dot{-} y = 0 \\
x = y \equiv x \leq y \wedge y \leq x &\equiv (x \dot{-} y) + (y \dot{-} x) = 0 \\
x \neq y \equiv \neg(x = y) &\equiv 1 \dot{-} ((x \dot{-} y) + (y \dot{-} x)) = 0
\end{aligned}$$

Satz 12 (S. 7). *Wenn eine Klasse $\mathcal{X}$ die Funktionen $S(x)$, $x + y$ und $x \dot{-} y$ enthält und abgeschlossen ist unter Substitution, dann ist die Menge der Relationen von $\mathcal{X}$ abgeschlossen unter den Operationen der Aussagenlogik.*

Beweis. Wenn $R(\vec{x})$ und $T(\vec{y})$ Relationen der Klasse $\mathcal{X}$ sind, gibt es Funktionen $r(\vec{x}), t(\vec{y}) \in \mathcal{X}$, sodass für alle $\vec{x}$ und alle $\vec{y}$ gilt:

$$R(\vec{x}) \equiv r(\vec{x}) = 0 \quad \text{und} \quad T(\vec{y}) \equiv t(\vec{y}) = 0$$

In den natürlichen Zahlen gilt außerdem für alle $a, b \in \mathbb{N}$:

$$1 \dot{-} a = 0 \equiv a \neq 0 \quad \text{und} \quad a + b = 0 \equiv a = 0 \wedge b = 0$$

Daraus folgt:

$$\begin{aligned}
\neg R(\vec{x}) &\equiv r(\vec{x}) \neq 0 &&\equiv 1 \dot{-} r(\vec{x}) = 0 \\
R(\vec{x}) \wedge T(\vec{y}) &\equiv r(\vec{x}) = 0 \wedge t(\vec{y}) = 0 &&\equiv r(\vec{x}) + t(\vec{y}) = 0
\end{aligned}$$

Die Funktion $r(\vec{x}) + t(\vec{y})$ entsteht durch Substitution aus der Addition. $1 \dot{-} r(\vec{x})$ benötigt die Subtraktion und den Nachfolger von 0. Folglich sind beide Funktionen in $\mathcal{X}$ und damit sind $R \wedge T$ sowie $\neg R$ Relationen von $\mathcal{X}$. Da sich alle aussagenlogischen Formeln aus $\wedge$ und $\neg$ bilden lassen, ist der Satz damit bewiesen. □

Korollar 13. *Anstelle der Funktion $x + y$ kann auch die Funktion $\sigma(x, y) = x * (1 \dot{-} y)$ verwendet werden.*

Beweis. In den natürlichen Zahlen gilt für alle $a, b \in \mathbb{N}$:

$$\begin{aligned}
a = 0 \wedge b = 0 &\equiv (1 \dot{-} a) * (1 \dot{-} b) \neq 0 \\
\Rightarrow R(\vec{x}) \wedge T(\vec{y}) &\equiv 1 \dot{-} \sigma(1 \dot{-} r(\vec{x}), t(\vec{y})) = 0
\end{aligned}$$ □

2.2 Die Klasse der elementaren Funktionen

Die Klasse der elementaren Funktionen wurde zuerst eingeführt vom ungarischen Mathematiker László Kalmár im Jahr 1951.

Definition 14 (S. 8). Die **Klasse der elementaren Funktionen** ist:

$$\mathcal{E} = \left[S(x), x + y, x \dot{-} y\,;\, \mathit{Sub}, \mathit{lim.}\sum, \mathit{lim.}\prod\right]$$

Beispiel 14.1 (S. 9). $x * y$, x^y und $x!$ sind elementare Funktionen.

Beweis. Die drei Funktionen lassen sich darstellen durch:

$$\begin{aligned} x * y &= \sum_{i \le y}(x + i \dot{-} i) \\ f(x,y) = x^{y+1} &= \prod_{i \le y}(x + i \dot{-} i) \\ x^y &= (1 \dot{-} y) + (1 \dot{-} (1 \dot{-} y)) * f(x, y \dot{-} 1) \\ x! &= \prod_{i \le (x \dot{-} 1)} (i + 1) \end{aligned}$$

□

Weitere elementare Funktionen lassen sich einfacher beschreiben, wenn man dazu weitere Operationen definiert, deren Abschluss die elementaren Funktionen nicht verlässt.

Definition 15 (S. 9). Eine Funktionsklasse heißt abgeschlossen unter der **Operation des beschränkten Minimums**, wenn sie für jede Funktion $f(\vec{u}, x)$ auch die Funktion $g(\vec{u}, y) =$

$$\mu(x \le y)\,[f(\vec{u}, x) = 0] = \begin{cases} \min\{x \mid x \le y \wedge f(\vec{u}, x) = 0\}, \text{ falls es existiert} \\ 0, \text{ sonst} \end{cases}$$

enthält. Falls es also kein $x \le y$ mit $f(\vec{u}, x) = 0$ gibt, ist $g(\vec{u}, y) = 0$.
Wenn die Operation $\mu(x \le y)\,[R(\vec{u}, x)]$ auf einer Relation angewendet wird, ist damit $\mu(x \le y)\,[r(\vec{u}, x) = 0]$ für eine beliebige Funktion mit $R(\vec{u}, x) \equiv r(\vec{u}, x) = 0$ gemeint.

Satz 16 (S. 9). *Die Klasse der elementaren Funktionen ist abgeschlossen unter der Operation des beschränkten Minimums.*

Beweis. Für alle $f(\vec{u}, x) \in \mathcal{E}$ und $g(\vec{u}, y) = \mu(x \le y)\,[f(\vec{u}, x) = 0]$ ist

$$h(\vec{u}, x) = \sum_{i \le x}(1 \dot{-} f(\vec{u}, i)) = |\{i \mid i \le x \wedge f(\vec{u}, i) = 0\}|$$

die Anzahl der $i \leq x$ mit $f(\vec{u}, i) = 0$ bzw. $1 \dot{-} f(\vec{u}, i) = 1$. Des Weiteren ist

$$h'(\vec{u}, y) = \sum_{i \leq y} (1 \dot{-} h(\vec{u}, i)) = |\{i \mid, i \leq y \wedge \forall x : x \leq i \rightarrow f(\vec{u}, x) \neq 0\}|$$

die Anzahl der $i \leq y$, die kleiner als das gesuchte Minimum sind, falls dieses existiert. Denn **wenn $m \leq y$ das Minimum ist**, gilt für alle $i \in \mathbb{N}$:

$$\begin{aligned} i < m \rightarrow f(\vec{u}, i) \neq 0 \quad &\Rightarrow \quad i < m \rightarrow h(\vec{u}, i) = 0 \\ f(\vec{u}, m) = 0 \quad &\Rightarrow \quad i \geq m \rightarrow h(\vec{u}, i) \neq 0 \end{aligned}$$

Mit $0 \in \mathbb{N}$ ist die Anzahl der Zahlen, die kleiner als das Minimum sind, genau das Minimum. **Falls das Minimum nicht existiert**, ist $f(\vec{u}, x) \neq 0$ für alle $x \leq y$. Daraus folgt:

$$h(\vec{u}, y) = 0 \quad \Leftrightarrow \quad h'(\vec{u}, y) = y + 1 \quad \Leftrightarrow \quad 1 \dot{-} (h'(\vec{u}, y) \dot{-} y) = 0$$

Demnach lässt sich die Funktion $g(\vec{u}, y)$ ausdrücken durch:

$$g(\vec{u}, y) = h'(\vec{u}, y) * (1 \dot{-} (h'(\vec{u}, y) \dot{-} y))$$

Da alle diese Funktionen durch Substitution oder beschränkte Addition aus elementaren Funktionen entstehen, ist $g(\vec{u}, y)$ ebenfalls elementar. □

Korollar 17 (S. 10). *Wenn eine Funktionsklasse $\mathcal{X}$ die Funktionen $x \dot{-} y$ und $\sigma(x, y) = x * (1 \dot{-} y)$ enthält und abgeschlossen ist unter Substitution und unter einer eingeschränkten Variante der beschränkten Addition, die einer Funktion $f(\vec{u}, i)$ die Funktion $g(\vec{u}, x) = \sum_{i \leq x} 1 \dot{-} f(\vec{u}, i)$ zuordnet, dann ist $\mathcal{X}$ abgeschlossen unter der Operation des beschränkten Minimums.*

Beweis. Der Beweis von Satz 16 verwendet neben der Substitution nur diese Funktionen und die Variante der beschränkten Addition. □

Definition 18 (S. 10). Die Relationen der Klasse der elementaren Funktionen heißen **elementare Relationen**.

Definition 19 (S. 10). Eine Relationsmenge über $\mathbb{N}$ heißt abgeschlossen unter der **Operation des beschränkten Existenzquantors**, wenn sie für jede Relation $R(\vec{u}, x)$ auch die Relation

$$T(\vec{u}, y) = \bigvee_{x \leq y} R(\vec{u}, x) \quad \text{enthält.}$$

Definition 20 (S. 11). Eine Relationsmenge über $\mathbb{N}$ heißt abgeschlossen unter der **Operation des beschränkten Allquantors**, wenn sie für jede Relation $R(\vec{u}, x)$ auch

die Relation

$$T(\vec{u}, y) = \bigwedge_{x \le y} R(\vec{u}, x) \quad \text{enthält.}$$

Satz 21 (S. 11). *Die elementaren Relationen sind abgeschlossen unter beiden Operationen der beschränkten Quantoren.*

Beweis. Für jede elementare Relation $R(\vec{u}, x)$ existiert eine elementare Funktion f mit $R(\vec{u}) \equiv f(\vec{u}, x) = 0$. Daraus lassen sich die elementaren Funktionen

$$g(\vec{u}, y) = \prod_{x \le y} f(\vec{u}, x) \quad \text{und} \quad h(\vec{u}, y) = \sum_{x \le y} f(\vec{u}, x) \quad \text{bilden, sodass}$$
$$\bigvee_{x \le y} R(\vec{u}, x) \equiv g(\vec{u}, y) = 0 \quad \text{und} \quad \bigwedge_{x \le y} R(\vec{u}, x) \equiv h(\vec{u}, y) = 0\,. \qquad \square$$

Satz 22 (S. 11). *Wenn eine Klasse $\mathcal{X}$ abgeschlossen ist unter Substitution und der Operation des beschränkten Minimums, dann sind die Relationen von $\mathcal{X}$ abgeschlossen unter der Operation des beschränkten Existenzquantors.*
Wenn die Relationen von $\mathcal{X}$ zusätzlich abgeschlossen sind unter den Operationen der Aussagenlogik, dann sind sie ebenfalls abgeschlossen unter der Operation des beschränkten Allquantors.

Beweis. Für jede Relation R von $\mathcal{X}$ mit $R(\vec{u}, x) \equiv f(\vec{u}, x) = 0$ gibt es

$$g(\vec{u}, y) = f\Big(\vec{u}, \mu(x \le y)\,[f(\vec{u}, x) = 0]\Big) \quad \in \mathcal{X}.$$

$$\Rightarrow g(\vec{u}, y) = 0 \equiv \bigvee_{x \le y} R(\vec{u}, x)$$

Wenn zusätzlich die Operationen der Aussagenlogik zur Verfügung stehen, dann gilt nach den Gesetzen von De-Morgan:

$$\bigwedge_{x \le y} R(\vec{u}, x) \equiv \neg \bigvee_{x \le y} \neg R(\vec{u}, x) \qquad \square$$

Lemma 23 (S. 13). *$x < y$, $x \le y$, $x = y$, $x \ne y$, $x \mid y$ und $x \in \mathbb{P}$ sind elementare Relationen. Dabei ist $\mathbb{P}$ die Menge aller Primzahlen.*

Beweis. $x \le y \equiv x \dot{-} y = 0$ und $x < y \equiv x + 1 \dot{-} y = 0$ werden durch elementare Funktionen beschrieben und die übrigen Relationen entstehen daraus durch Operationen der Aussagenlogik, beschränkte Quantoren und Substitution:

$$x = y \;\equiv\; x \le y \wedge y \le x \qquad x \ne y \;\equiv\; \neg(x = y)$$
$$x \mid y \;\equiv \bigvee_{i \le y} (i * x = y) \qquad x \in \mathbb{P} \;\equiv \bigwedge_{i \le x} (i \mid x \to i = 1 \vee i = x) \qquad \square$$

Beispiel 23.1 (S. 12). Mit der Ordnungsrelation und der Operation des beschränkten Minimums lässt sich Division auf den natürlichen Zahlen definieren durch:

$$\left\lfloor \frac{x}{y} \right\rfloor = \mu(i \leq x)\,[x + 1 \leq i * (y + 1)]$$
$$\Rightarrow x \bmod y = x \dot{-} y * \left\lfloor \frac{x}{y} \right\rfloor$$

In den folgenden Abschnitten werden neben dem beschränkten Minimum noch zwei Maximumsoperationen auf elementaren Funktionen auftauchen.

Definition 24 (S. 12). Aus einer Relation $R(\vec{u}, x)$ entsteht durch die **Operation des beschränkten Maximums** die Funktion

$$\max(x \leq y)\,[R(\vec{u}, x)] = \mu(x \leq y)\left[R(\vec{u}, x) \wedge \bigwedge_{i \leq y} (R(\vec{u}, i) \rightarrow i \leq x)\right],$$

die das größte $x \leq y$ berechnet, für welches $R(\vec{u}, x)$ wahr ist.

Definition 25 (S. 13). Aus einer Funktion $f(\vec{u}, x)$ entsteht durch die **Operation des beschränkten Maximalwertes** die Funktion

$$\max_{x \leq y} \{f(\vec{u}, x)\} = f\left(\vec{u},\ \mu(x \leq y)\left[\bigwedge_{i \leq y} f(\vec{u}, i) \leq f(\vec{u}, x)\right]\right),$$

die den größten Wert von $f(\vec{u}, x)$ für $x \leq y$ berechnet.

Anhand der Definitionen wird deutlich, dass die elementaren Funktionen auch unter diesen beiden Operationen abgeschlossen sind.

2.2.1 Primzahlen und Rekursion

Wir betrachten nun die Menge der Primzahlen $\mathbb{P}$ im Zusammenhang mit der Klasse $\mathcal{E}$. Die Zerlegung von natürlichen Zahlen in ihre Primfaktoren gilt im Allgemeinen als schwieriges Problem und ein Großteil der heute verwendeten Kryptographieverfahren beruht darauf. Wir werden nun zeigen, dass elementare Funktionen mächtig genug sind, um Primfaktoren zu berechnen.

Definition 26 (S. 13). Um alle Primzahlen zu berechnen definieren wir die folgenden elementaren Funktionen:

$$\operatorname{dexp}(x, y) = \max(i \leq x) \left[y^i \mid x \right]$$

(Der größte Exponent i, sodass y^i noch x teilt)

$$\operatorname{Prim}(x, k) \equiv x \in \mathbb{P} \wedge \sum_{y \leq x} (1 \dot{-} [y \in \mathbb{P}]) = k + 1$$
$$\text{mit } [y \in \mathbb{P}] = 0 \equiv y \in \mathbb{P}$$

(Die Relation „x ist die k-te Primzahl“, sodass $\operatorname{Prim}(2, 0)$ gilt)

$$\operatorname{prim}(k) = \mu(x \leq (k+2)^2) \left[\operatorname{Prim}(x, k) \right]$$

(Die Funktion, die die k-te Primzahl berechnet)

$$\operatorname{prex}(x, k) = \operatorname{dexp}(x, \operatorname{prim}(k))$$

(Der Exponent von $\operatorname{prim}(k)$ in der Primfaktorzerlegung von x)

Die Funktion $\operatorname{prim}(k)$ benötigt in ihrer Definition das zahlentheoretische Resultat $\operatorname{prim}(k) \leq (k+2)^2$, welches hier nicht bewiesen wird. Grzegorczyk verwendete an dieser Stelle die Funktion $(k+2)^{k+2}$, die zwar elementar, aber auch exponentiell ist. Dieser Unterschied wird im dritten Kapitel noch eine Rolle spielen.
Die Folge der Primzahlen lässt sich nun dazu verwenden, endliche Sequenzen von Zahlen zu kodieren.

Lemma 27 (S. 14). *Für jede Relation $R(x_0, \dots, x_n)$ von $n+1$ natürlichen Zahlen gilt:*

$$\exists x_0 \dots \exists x_n : R(x_0, \dots, x_n) \equiv \exists x : R(\operatorname{prex}(x, 0), \dots, \operatorname{prex}(x, n))$$

Beweis. Wenn es natürliche Zahlen $x_0, \dots, x_n$ gibt, für die $R(x_0, \dots, x_n)$ wahr ist, dann gibt es $x = \prod_{k=0}^{n} \operatorname{prim}(k)^{x_k}$. Nach dem Fundamentalsatz der Algebra ist dies die eindeutige Primfaktorzerlegung von x. Daraus folgt:

$$\forall k \leq n : \operatorname{prex}(x, k) = x_k$$
$$\Rightarrow R(\operatorname{prex}(x, 0), \dots, \operatorname{prex}(x, n))$$

Wenn es wiederum ein x mit $R(\operatorname{prex}(x, 0), \dots, \operatorname{prex}(x, n))$ gibt, sind $x_0, \dots, x_n$ gegeben durch $x_k = \operatorname{prex}(x, k)$ für $1 \leq k \leq n$. □

Definition 28 (S. 14). Eine Funktionsklasse heißt abgeschlossen unter der **Operation der beschränkten Rekursion**, wenn sie für alle Funktionen $f(\vec{u})$, $g(\vec{u}, x, y)$ und $h(\vec{u}, x)$ auch die Funktion $F(\vec{u}, x)$ enthält, sodass für alle $\vec{u}$ und x gilt:

$$F(\vec{u}, 0) = f(\vec{u}) \tag{a}$$
$$F(\vec{u}, x+1) = g(\vec{u}, x, F(\vec{u}, x)) \tag{b}$$
$$F(\vec{u}, x) \leq h(\vec{u}, x) \tag{c}$$

Im folgenden wird diese Operation abgekürzt durch *lim.Rec.*

Die Funktion F wird also durch Rekursion auf der letzten Variablen definiert mit einem Startwert (a) und einer Rekursionsvorschrift (b). Bei der beschränkten Rekursion darf die Funktion allerdings nicht größer werden als eine Schrankenfunktion (c).

Satz 29 (S. 16). *Die Klasse der elementaren Funktionen ist abgeschlossen unter der Operation der beschränkten Rekursion.*

Beweis. Seien $f, g, h \in \mathcal{E}$ und F Funktionen, die die Bedingungen a, b und c aus Definition 28 erfüllen. Dann ist $y = f(\vec{u}, x)$ äquivalent dazu, dass es eine Sequenz von $x+1$ Zahlen $a_0, \ldots, a_x$ gibt, für die gilt:

$$\begin{aligned} a_0 &= f(\vec{u}, 0) \\ \forall\, i < x : a_{i+1} &= g(\vec{u}, i, a_i) \\ a_x &= y \end{aligned}$$

Da die Länge der Sequenz durch x gegeben ist, können wir nun Lemma 27 anwenden, um eine Relation $R(\vec{u}, x, y)$ aus diesen Bedingungen zu formulieren, mit deren Hilfe wir nach dem richtigen y suchen können. Dazu ersetzen wir also jedes a_i durch $\mathrm{prex}(a, i)$ und erhalten dadurch:

$$\begin{aligned} \mathrm{prex}(a, 0) &= f(\vec{u}, 0) \\ \forall\, i < x : \mathrm{prex}(a, i+1) &= g(\vec{u}, i, \mathrm{prex}(a, i)) \\ \mathrm{prex}(a, x) &= y \end{aligned}$$

Daraus lässt sich eine elementare Relation bilden:

$$R(\vec{u}, x, y) = \bigvee_{a \leq m(\vec{u}, x)} \left(\begin{array}{c} \mathrm{prex}(a, 0) = f(\vec{u}, 0) \wedge \\ \bigwedge\limits_{i \leq x \dot{-} 1} \Big(\mathrm{prex}(a, i+1) = g(\vec{u}, i, \mathrm{prex}(a, i)) \Big) \\ \wedge\, \mathrm{prex}(a, x) = y \end{array} \right)$$

An dieser Stelle fehlt noch eine obere Schranke für den Existenzquantor $m(\vec{u}, x)$. Wir wissen aber aufgrund der Schrankenfunktion für alle $i \leq x$:

$$\begin{aligned} a_i = F(\vec{u}, i) &\leq h(\vec{u}, i) \\ \Rightarrow a_i &\leq \max_{j \leq x} \{h(\vec{u}, j)\} \end{aligned}$$

Für das kleinste mögliche $a = \prod_{i \leq x} \text{prim}(i)^{a_i}$ folgt daraus:

$$a \leq \prod_{i \leq x} \text{prim}(i)^{\max_{j \leq x}\{h(\vec{u},j)\}} \leq \text{prim}(x)^{(x+1) * \max_{j \leq x}\{h(\vec{u},j)\}} = m(\vec{u}, x)$$

Dies funktioniert, da $\text{prim}(x)$ der größte Primfaktor von a ist und damit haben wir eine offensichtlich elementare Funktion für m gefunden.
Da der Wert von $F(\vec{u}, x)$ durch $h(\vec{u}, x)$ beschränkt ist, kann er nun mit Hilfe des beschränkten Minimums berechnet werden:

$$F(\vec{u}, x) = \quad \mu(y \leq h(\vec{u}, x))\,[R(\vec{u}, x, y)] \quad \in \mathcal{E} \qquad \square$$

2.2.2 Äquivalente Definition der Klasse $\mathcal{E}$

Satz 30 (vgl. S. 17-18). *Die elementaren Funktionen lassen sich jetzt auf andere Weise charakterisieren:*

$$\mathcal{E} = [S(x), x^y\,;\, Sub, lim.Rec]$$

Beweis.[1] Wir definieren $\mathcal{E}' = [S(x), x^y\,;\, Sub, lim.Rec]$.
Die Inklusion $\mathcal{E}' \subseteq \mathcal{E}$ ist offensichtlich, da x^y elementar ist und Satz 29 gilt. Die zwei weiteren Initialfunktionen von $\mathcal{E}$ und $x * y$ lassen sich in $\mathcal{E}'$ wie folgt bilden:

$$\begin{aligned} x + 0 &= x^{0+1} = x & x * 0 &= 0^{0+1} = 0 \\ x + (y+1) &= (x+y) + 1 & x * (y+1) &= (x * y) + x \\ x + y &\leq (x+1+1)^{y+1} & x * y &\leq (x+1)^y \end{aligned}$$

$$\begin{aligned} 0 \dot{-} 1 &= 0 + 0 & x \dot{-} 0 &= x + 0 \\ (x+1) \dot{-} 1 &= (x \dot{-} 1) & x \dot{-} (y+1) &= (x \dot{-} y) \dot{-} 1 \\ x \dot{-} 1 &\leq x + 0 & x \dot{-} y &\leq x + 0 \end{aligned}$$

[1] Grzegorczyk hat diesen Beweis zweigeteilt über einen Zwischenschritt mit beschränktem Minimum.

Für jede Funktion $f(\vec{u}, i) \in \mathcal{E}'$ enthält $\mathcal{E}'$ auch die Funktionen g und h mit:

$$\begin{aligned} g(\vec{u}, 0) &= f(\vec{u}, 0) & h(\vec{u}, 0) &= f(\vec{u}, 0) \\ g(\vec{u}, x+1) &= f(\vec{u}, x+1) + g(\vec{u}, x) & h(\vec{u}, x+1) &= f(\vec{u}, x+1) * h(\vec{u}, x) \\ g(\vec{u}, x) &\leq \max_{i \leq x} \{f(\vec{u}, i)\} * (x+1) & h(\vec{u}, x) &\leq \max_{i \leq x} \{f(\vec{u}, i)\}^{x+1} \\ \Rightarrow g(\vec{u}, x) &= \sum_{i \leq x} f(\vec{u}, i) & \Rightarrow h(\vec{u}, x) &= \prod_{i \leq x} f(\vec{u}, i) \end{aligned}$$

Folglich ist $\mathcal{E}'$ ebenfalls abgeschlossen unter der Operation der beschränkten Addition und der Operation der beschränkten Multiplikation. Es gilt also:

$$\mathcal{E} = \left[S(x), x + y, x \dot{-} y\,;\, \mathit{Sub}, \mathit{lim.}\sum, \mathit{lim.}\prod\right] \subseteq \mathcal{E}' \qquad \square$$

2.2.3 Paarfunktionen

Eine Möglichkeit, Paare von Zahlen zu kodieren, ist:

$$f(x, y) = 2^x * 3^y, \quad g(z) = \text{prex}(z, 0) \quad \text{und} \quad h(z) = \text{prex}(z, 1)$$

Die Werte von f enthalten als Primfaktoren nur 2 und 3 und werden dadurch schon bei kleinen x, y sehr groß. Wenn man jedoch keine beliebig langen Folgen kodieren möchte, ist es nicht zwingend erforderlich, auf Primzahlen zurückzugreifen.

Definition 31 (S. 4). Ein Tripel aus drei Funktionen $I(x, y)$, $K(z)$, $L(z)$ enthält Paarfunktionen, wenn für alle natürlichen Zahlen x, y, z gilt:

$$\begin{aligned} I(K(z), L(z)) &= z \\ K(I(x, y)) &= x \\ L(I(x, y)) &= y \end{aligned}$$

Um das zu erreichen, muss $I(x, y)$ bijektiv sein. $K(z)$ und $L(z)$ dagegen müssen zwar surjektiv, aber nicht injektiv sein.

Satz 32 (S. 4). *Ein mögliches Tripel von Paarfunktionen ist:*

$$\begin{aligned} I(x, y) &= (x + y)^2 + x \\ K(z) &= z - \lfloor \sqrt{z} \rfloor^2 \\ L(z) &= \lfloor \sqrt{z} \rfloor - K(z) \end{aligned}$$

Beweis. Für alle $x, y \in \mathbb{N}$ ist $\lfloor\sqrt{I(x,y)}\rfloor = x + y$, denn es gilt:

$$\begin{aligned}
\sqrt{I(x,y)} = \sqrt{(x+y)^2 + x} &\geq \sqrt{(x+y)^2} && = x + y \\
\sqrt{I(x,y)} = \sqrt{(x+y)^2 + x} &< \sqrt{(x+y)^2 + 2x + 2y + 1} \\
&= \sqrt{x^2 + 2x + 2xy + y^2 + 2y + 1} \\
&= \sqrt{x^2 + 2(x(y+1)) + (y+1)^2} && = x + y + 1
\end{aligned}$$

Daraus folgt für alle $x, y, z \in \mathbb{N}$:

$$\begin{aligned}
I(K(z), L(z)) &= (K(z) + L(z))^2 + K(z) \\
&= (K(z) + \lfloor\sqrt{z}\rfloor - K(z))^2 + K(z) \\
&= \lfloor\sqrt{z}\rfloor^2 + z - \lfloor\sqrt{z}\rfloor^2 && = z \\
K(I(x,y)) &= I(x,y) - \lfloor\sqrt{I(x,y)}\rfloor^2 \\
&= (x+y)^2 + x - (x+y)^2 && = x \\
L(I(x,y)) &= \lfloor\sqrt{I(x,y)}\rfloor - K(I(x,y)) \\
&= (x+y) - x && = y
\end{aligned}$$

□

Im Verlauf dieser Arbeit stehen I, K und L immer für diese drei Funktionen.
Sobald man eine Funktion hat, die ein Paar von zwei Zahlen kodiert, kann man eine Folge von Funktionen definieren, um beliebig große Tupel zu kodieren. Dazu verwenden wir die folgende rekursive Definition:

Definition 33 (vgl. S. 21).[2] Für alle n sind Tupelfunktionen gegeben durch:

$$\begin{aligned}
T_1(x_1) &= x_1 \\
\text{Für } n > 1 :\ T_n(x_1, \ldots, x_n) &= I(T_{n-1}(x_1, \ldots, x_{n-1}), x_n)
\end{aligned}$$

$$\begin{aligned}
C_1^1(x) &= x \\
\text{Für } n > 1 :\ C_n^n(x) &= L(x) \\
\text{und für } 0 < i < n :\ C_i^n(x) &= C_i^{n-1}(K(x))
\end{aligned}$$

Für $n > 1$ entstehen alle diese Funktionen durch Substitution aus I, K und L. Außerdem

[2] Grzegorczyk hat die Tupel nicht links-, sondern rechtsrekursiv definiert, also mit $T_n(x_1, \ldots, x_n) = I(x_1, T_{n-1}(x_2, \ldots, x_n))$. Außerdem hat er den letzten Wert im Tupel immer gesondert betrachtet, was meiner Meinung nach den folgenden Beweis deutlich verkompliziert hat.

gilt für alle $n > 0$ und alle $0 < i \leq n$:

$$T_n(C_1^n(x), \ldots, C_n^n(x)) = x$$
$$C_i^n(T_n(x_1, \ldots, x_n)) = x_i$$

Beispiel 33.1. Für $n = 4$ sind die Tupelfunktionen gegeben durch:

$$T_4(x_1, x_2, x_3, x_4) = I(I(I(x_1, x_2), x_3), x_4)$$

$$C_1^4(x) = K(K(K(x))) \qquad C_2^4(x) = L(K(K(x)))$$
$$C_3^4(x) = L(K(x)) \qquad C_4^4(x) = L(x)$$

2.3 Allgemeinere Funktionsklassen

In diesem Abschnitt werden einige allgemeinere Aussagen über Funktionsklassen bewiesen, die dann im nächsten Abschnitt verwendet werden.

2.3.1 Einstellige Funktionsklassen

Definition 34 (S. 5). Für eine Funktionsklasse $\mathcal{X}$ sei $\mathcal{X}_n$ die Teilklasse der Funktionen mit n Parametern.

Satz 35 (vgl. S. 20).[3] *Ist eine Funktionsklasse* $\mathcal{X} = [I(x, y), K(z), L(z), f_1, \ldots, f_n ; Sub]$, *dann gilt:*

$$\mathcal{X}_1 = [I(x, x), I(x, 0), I(0, x), K(z), L(z), f_1', \ldots, f_n' ; \circ, \mathbf{O}_{ILK}]$$

Dabei ist $\circ$ *die Komposition von Funktionen mit einem Argument und* $\mathbf{O}_{ILK}$ *ist die Operation, die zu jeder Funktion* $f(x)$ *die Funktionen*

$$g(x) = I\Big(f\Big(K(x)\Big), L(x)\Big) \quad \textit{und} \quad h(x) = I\Big(K(x), f\Big(L(x)\Big)\Big)$$

erzeugt. Aus jedem $f_i(x_1, \ldots, x_m)$ *mit beliebig vielen Argumenten entsteht:*

$$f_i'(x) = f(C_1^m(x), \ldots, C_m^m(x))$$

[3] Im Originalsatz ist $\mathcal{X}$ noch abgeschlossen unter beliebigen Operationen, die die Zahl der Parameter nicht erhöhen, aber diese werden hier nicht benötigt. Die Formulierung „induktiv definierbar“ ist äquivalent zu unserem algebraischen Hüllenoperator.

Beweis.[4] Wir definieren zunächst:

$$\mathcal{X}_1' = [I(x,x)\,, I(x,0)\,, I(0,x)\,, K(z)\,, L(z)\,, f_1', \ldots, f_n'\,;\, \circ, \mathbf{O}_{ILK}]$$

Alle Initialfunktionen von $\mathcal{X}_1'$ entstehen durch Substitution aus den Initialfunktionen von $\mathcal{X}_1$. Auch die beiden Operationen lassen sich mittels Substitution umsetzen, wenn mehrere Argumente zur Verfügung stehen. Also gilt:

$$\mathcal{X}_1' \subseteq \mathcal{X}_1$$

Für die entgegengesetzte Richtung benötigen wir die Tupelfunktionen. $\mathcal{X}_1'$ enthält die Identitätsfunktion $id(x) = K(I(x,0))$ und damit alle $C_i^m(x)$, da diese durch Komposition aus id, K und L entstehen. Außerdem enthält $\mathcal{X}_1'$ die Nullfunktion $Z(x) = 0 = L(I(x,0))$. Des Weiteren gilt:

Lemma 36. *Für alle Funktionen $g(x), h(x) \in \mathcal{X}_1'$ enthält $\mathcal{X}_1'$ die Funktion:*

$$F(x) = I(g(x), h(x))$$

Beweis. Nach der Operation $\mathbf{O}_{ILK}$ enthält $\mathcal{X}_1'$ die Funktionen $I(g(K(x)), L(x))$ und $I(K(x)\,, h(L(x)))$. Daraus folgt durch Komposition:

$$\begin{aligned} I\Big(g\Big(K(I(x,x))\Big), L(I(x,x))\Big) &= I(g(x),x) && \in \mathcal{X}_1' \\ I\Big(K(I(g(x),x))\,, h\Big(L(I(g(x),x))\Big)\Big) &= I(g(x),h(x)) && \in \mathcal{X}_1' \end{aligned}$$ □

Lemma 37. *Für alle Funktionen $g_1(x), \ldots, g_m(x) \in \mathcal{X}_1'$ enthält $\mathcal{X}_1'$ auch:*

$$g(x) = T_m(g_1(x), \ldots, g_m(x))$$

Beweis. **Induktion** über m:

Für $m = 1$ ist $g(x) = T_1(g_1(x)) = g_1(x)$.

Falls $T_{m-1}(g_1(x), \ldots, g_{m-1}(x))$ in $\mathcal{X}_1'$ enthalten ist, folgt daraus

$$g(x) = T_m(g_1(x), \ldots, g_m(x)) = I(T_{m-1}(g_1(x), \ldots, g_{m-1}(x)), g_m(x)) \in \mathcal{X}_1'$$

unter Verwendung von Lemma 36. □

[4] Wie bereits erwähnt, verläuft dieser Beweis leicht anders als bei Grzegorczyk aufgrund der angepassten Definition 33.

Lemma 38. *Für alle $F(x_1, \dots, x_m) \in \mathcal{X}$ enthält $\mathcal{X}_1'$ die Funktion*

$$F'(x) = F(C_1^m(x), \dots, C_m^m(x)).$$

Beweis. **Induktion** über die Ordnung von f:

Für die Initialfunktionen von $\mathcal{X}$ gilt:

$$I\Big(C_1^2(x), C_2^2(x)\Big) = I(K(x), L(x)) = id(x) \in \mathcal{X}_1'$$
$$K\Big(C_1^1(x)\Big) = K(x) \in \mathcal{X}_1' \qquad L\Big(C_1^1(x)\Big) = L(x) \in \mathcal{X}_1' \qquad f_i' \in \mathcal{X}_1'$$

Wir treffen nun die Annahme, dass für alle Funktionen g, h mit Ordnung kleiner als k die $g'(x), h'(x) \in \mathcal{X}_1'$ existieren. Diese sind dann:

$$g'(x) = g(C_1^s(x), \dots, C_s^s(x))$$
$$h'(x) = h(C_1^t(x), \dots, C_t^t(x))$$

Wenn F durch Substitution von h aus g entsteht, dann gilt:

$$F(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_s, y_1, \dots, y_t) = g(x_1, \dots, x_{i-1}, h(y_1, \dots, y_t), x_{i+1}, \dots, x_s)$$

$$\begin{aligned} F'(x) &= F(C_1^m(x), \dots, C_m^m(x)) \\ &= g(C_1^m(x), \dots, C_{i-1}^m(x), h(C_s^m(x), \dots, C_m^m(x)), C_i^m(x), \dots, C_{s-1}^m(x)) \\ &= g'(T_s(C_1^m(x), \dots, C_{i-1}^m(x), h(C_s^m(x), \dots, C_m^m(x)), C_i^m(x), \dots, C_{s-1}^m(x))) \end{aligned}$$

Nach Lemma 37 ist $T_t(C_s^m(x), \dots, C_m^m(x)) \in \mathcal{X}_1'$, also folgt daraus:

$$\begin{aligned} & h(C_s^m(x), \dots, C_m^m(x)) = h'(T_t(C_s^m(x), \dots, C_m^m(x))) \in \mathcal{X}_1' \\ \Rightarrow\ & T_s(C_1^m(x), \dots, C_{i-1}^m(x), h(...), C_i^m(x), \dots, C_{s-1}^m(x)) \in \mathcal{X}_1' \\ \Rightarrow\ & F'(x) = g'(T_s(...)) \in \mathcal{X}_1' \end{aligned}$$

Wenn F durch Identifikation von Variablen aus g entsteht, dann gilt:

$$F(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_{j-1}, x_{j+1}, \dots, x_s, y) = g(x_1, \dots, x_{i-1}, y, x_{i+1}, \dots, x_{j-1}, y, x_{j+1}, \dots, x_s)$$

$$F'(x) = g'(T_s(C_1^{s-1}(x), \dots, C_{i-1}^{s-1}(x), C_{s-1}^{s-1}(x), C_i^{s-1}(x), \dots, C_{j-2}^{s-1}(x), C_{s-1}^{s-1}(x), C_{j-1}^{s-1}(x), \dots, C_{s-2}^{s-1}(x)))$$

Wie oben ist die Komposition von g' mit dem Tupel wieder in $\mathcal{X}_1'$.

Wenn F durch Substitution von 0 aus g entsteht, dann gilt:

$$\begin{aligned} F(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_s) &= g(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_s) \\ &= g(x_1, \dots, x_{i-1}, Z(y), x_{i+1}, \dots, x_s) \text{ für alle } y \in \mathbb{N} \end{aligned}$$

$$F'(x) = g(C_1^m(x), \dots, C_{i-1}^m(x), Z(x), C_i^m(x), \dots, C_{s-1}^m(x)) \in \mathcal{X}_1'$$

Nach der **Induktion** gibt es also für jedes $F \in \mathcal{X}$ ein $F'(x) \in \mathcal{X}_1'$. □

Folglich gibt es auch für jedes $F(x) \in \mathcal{X}_1$ ein $F'(x) \in \mathcal{X}_1'$, sodass

$$F'(x) = F(C_1^1(x)) = F(x) \in \mathcal{X}_1',$$

also gilt $\mathcal{X}_1 \subseteq \mathcal{X}_1'$ und dies ist der Beweis für Satz 35. □

2.3.2 Dominante Funktionen

Definition 39 (S. 25). Eine Funktion $f(x_1, \dots, x_n)$ heißt **(streng) monoton wachsend** im k-ten Argument, wenn für alle $x_1, \dots, x_n$ gilt:

$$\begin{aligned} \forall y : y > x_k \rightarrow f(x_1, \dots, x_{k-1}, y, x_{k+1}, \dots, x_n) \geq f(x_1, \dots, x_n) \\ \text{bzw. } \forall y : y > x_k \rightarrow f(x_1, \dots, x_{k-1}, y, x_{k+1}, \dots, x_n) > f(x_1, \dots, x_n) \end{aligned}$$

Eine Funktion heißt **(streng) monoton wachsend**, wenn sie (streng) monoton wachsend in jedem Argument ist.

Definition 40 (S. 25). Eine Funktion $g(\vec{u})$ **dominiert** $f(\vec{u})$, wenn für alle $\vec{u}$ gilt:

$$f(\vec{u}) \leq g(\vec{u})$$

Definition 41 (S. 25). Eine Funktion $g(x)$ von einem Argument wächst schneller als $f(x)$, wenn es ein x_0 gibt, sodass für alle $x \geq x_0$ gilt:

$$f(x) < g(x)$$

Diese Aussage ist nicht äquivalent zu $f(x) \in \mathbf{o}(g(x))$.

Satz 42 (vgl. S. 25). *Sei $\mathcal{Y} = [f_1, \dots, f_n\,;\, Sub, lim.Rec]$ und $[f_1', \dots, f_n'\,;\, Sub] \subseteq \mathcal{X}$, sodass für alle $i \in \{1, \dots, n\}$ f_i' monoton wachsend ist und f_i dominiert. Dann wird jede Funktion $f \in \mathcal{Y}$ dominiert von einer monoton wachsenden Funktion $f' \in \mathcal{X}$.*

Beweis. **Induktion** über die Ordnung der Funktionen aus $\mathcal{Y}$:
Die Funktionen der Ordnung 0 sind $f_1, \ldots, f_n$ und werden dominiert von den monoton wachsenden $f_1', \ldots, f_n'$. Seien nun $g, h \in \mathcal{Y}$ Funktion mit Ordnung von höchstens n, die von $g', h' \in \mathcal{X}$ dominiert werden.

Falls $F \in \mathcal{Y}$ durch Substitution von h aus g entsteht, gilt:

$$F(\vec{u}, \vec{v}, \vec{y}) = g(\vec{u}, h(\vec{y}), \vec{v}) \leq g'(\vec{u}, h(\vec{y}), \vec{v})$$

Da $h(\vec{y}) \leq h'(\vec{y})$ und g' monoton wachsend, folgt daraus:

$$F(\vec{u}, \vec{v}, \vec{y}) \leq g'(\vec{u}, h'(\vec{y}), \vec{v}) \in \mathcal{X}$$

Als Komposition von monoton wachsenden Funktionen ist $g'(\vec{u}, h'(\vec{y}), \vec{v})$ ebenfalls monoton wachsend.

Falls $F \in \mathcal{Y}$ durch Identifikation von Variablen aus g entsteht, gilt:

$$F(\vec{u}, \vec{v}, \vec{w}, y) = g(\vec{u}, y, \vec{v}, y, \vec{w}) \leq g'(\vec{u}, y, \vec{v}, y, \vec{w}) \in \mathcal{X}$$

Falls $F \in \mathcal{Y}$ durch Identifikation von Variablen aus g entsteht, gilt:

$$F(\vec{u}, \vec{v}) = g(\vec{u}, 0, \vec{v}) \leq g'(\vec{u}, 0, \vec{v}) \in \mathcal{X}$$

Falls $F \in \mathcal{Y}$ durch beschränkte Rekursion mit Schrankenfunktion g entsteht, gilt:

$$F(\vec{u}, x) \leq g(\vec{u}, x) \leq g'(\vec{u}, x) \in \mathcal{X}$$

Insgesamt werden also alle Funktionen $F \in \mathcal{Y}$ der Ordnung $n+1$ von einer monoton wachsenden Funktion aus $\mathcal{X}$ dominiert. Folglich werden alle Funktionen aus $\mathcal{Y}$ von einer solchen Funktion dominiert. □

Satz 43 (vgl. S. 26).[5] *Wenn jede Funktion der Klasse $\mathcal{Y}$ von einer Funktion in $\mathcal{X}$ dominiert wird und die Funktion $f(x)$ schneller wächst als jede Funktion in $\mathcal{X}_1$, dann wächst $f(x)$ auch schneller als jede Funktion in $\mathcal{Y}_1$.*

Beweis. Für jede Funktion $g(x) \in \mathcal{Y}_1$ existiert $g'(x) \in \mathcal{X}_1$, sodass $\forall x : g(x) \leq g'(x)$. Da $f(x)$ schneller wächst als $g'(x) \in \mathcal{X}_1$, gibt es ein x_0, sodass für $x \geq x_0$ gilt:

$$g(x) \leq g'(x) < f(x).$$

Also wächst $f(x)$ auch schneller als $g(x) \in \mathcal{Y}_1$. □

[5] Grzegorczyks Satz ist hier etwas kompliziert, da er die Aussage des vorigen Satzes nochmal enthält.

2.4 Die Klassen $\mathcal{E}^n$

Grzegorczyk hat nun die Eigenschaften der elementaren Funktionen auf eine Reihe verschiedener Funktionsklassen übertragen und dadurch eine ganze Hierarchie geschaffen, die heute als **Grzegorczyk Hierarchie** bekannt ist.

2.4.1 Basisfunktionen

In diesem Abschnitt betrachten wir eine Folge von total berechenbaren Funktionen, die als Initialfunktionen der Klassen $\mathcal{E}^n$ dienen. Sei dazu:

$$f_0(x,y) = y+1 \qquad f_1(x,y) = x+y \qquad f_2(x,y) = (x+1)*(y+1)$$

$$\begin{aligned} f_{n+1}(0,y) &= f_n(y+1,y+1) \\ f_{n+1}(x+1,y) &= f_{n+1}(x, f_{n+1}(x,y)) \end{aligned}$$

Satz 44 (S. 28). *Für alle $n > 1$ gilt $f_n(x,y) > y$.*

Beweis. **Induktion** über n und über x:

Für $n = 2$ ist $f_2(x,y) = (x+1)*(y+1) > y$.

Wenn $f_n(x,y) > y$ für ein n gilt, dann folgt daraus:

$$f_{n+1}(0,y) = f_n(y+1,y+1) > y+1 > y$$

Aus $f_{n+1}(x,y) > y$ folgt:

$$f_{n+1}(x+1,y) = f_{n+1}(x, f_{n+1}(x,y)) > f_{n+1}(x,y) > y$$

Folglich gilt $f_{n+1}(x,y) > y$ für alle x.

Insgesamt gilt also $f_n(x,y) > y$ für alle $n \geq 2$. □

Satz 45 (S. 28). *Für alle $n > 0$ gilt $f_n(x+1,y) > f_n(x,y)$.*

Beweis. Für $n = 1$ gilt $(x+1)+y > x+y$.

Für $n = 2$ gilt $(x+2)*(y+1) > (x+1)*(y+1)$.

Für alle $n > 2$ gilt $f_n(x+1,y) = f_n(x, f_n(x,y)) > f_n(x,y)$ □

Satz 46 (S. 28). *Für alle $n > 0$ gilt $f_n(x, y+1) > f_n(x, y)$.*

Beweis. **Induktion** über n und x:

Für $n = 1$ gilt $x + (y+1) > x + y$.

Für $n = 2$ gilt $(x+1) * (y+2) > (x+1) * (y+1)$.

Wenn $f_n(x, y+1) > f_n(x, y)$ für ein n gilt, dann folgt daraus:

$$\begin{aligned} f_{n+1}(0, y+1) = f_n(y+2, y+2) &> f_n(y+1, y+2) \\ &> f_n(y+1, y+1) = f_n(0, y) \end{aligned}$$

Aus $f_{n+1}(x, y+1) > f_{n+1}(x, y)$ für alle y folgt:

$$\begin{aligned} f_{n+1}(x+1, y+1) = f_{n+1}(x, f_{n+1}(x, y+1)) &> f_{n+1}(x, f_{n+1}(x, y)) \\ &= f_{n+1}(x+1, y) \end{aligned}$$

Folglich gilt $f_{n+1}(x, y+1) > f_{n+1}(x, y)$ für alle x.

Insgesamt gilt also $f_n(x, y+1) > f_n(x, y)$ für alle $n \geq 1$. □

Also sind alle $f_n(x, y)$ streng monoton wachsend.

Satz 47 (vgl. S. 32). *Für alle $n > 0$ gilt $f_{n+1}(x, y) > f_n(x, y)$.*

Beweis. **Induktion** über n und x:

Für $n = 1$ gilt $f_2(x, y) = (x+1) * (y+1) > x + y = f_1(x, y)$.

Wenn $f_{n+1}(x, y) > f_n(x, y)$ für ein n gilt, dann folgt daraus, da alle $f_n(x, y)$ streng monoton wachsend sind:

$$f_{n+1}(0, y) = f_n(y+1, y+1) > f_n(0, y)$$

Aus $f_{n+1}(x, y) > f_n(x, y)$ für alle y folgt:

$$\begin{aligned} f_{n+1}(x+1, y) = f_{n+1}(x, f_{n+1}(x, y)) &> f_{n+1}(x, f_n(x, y)) \\ &> f_n(x, f_n(x, y)) = f_n(x+1, y) \end{aligned}$$

Folglich gilt $f_{n+1}(x, y+1) > f_{n+1}(x, y)$ für alle x.

Insgesamt gilt also $f_{n+1}(x, y) > f_n(x, y)$ für alle $n \geq 1$. □

Definition 48 (S. 29). Als letztes benötigen wir noch die Projektionsfunktionen:

$$p_k^n(x_1, \ldots, x_n) = x_k$$

2.4.2 Eigenschaften

Nun können wir diese Funkionen verwenden, um die Klassen der Grzegorczyk Hierarchie zu definieren, die auf Substitution und Rekursion basieren:

Definition 49 (S. 29). Die **Klassen der Grzegorczyk Hierarchie** sind:

$$\mathcal{E}^n = \left[S(x), p_1^2, p_2^2, f_n(x,y)\,;\, Sub, lim.Rec\right]$$

Satz 50 (S. 29). *$\mathcal{E}^3$ ist die Klasse der elementaren Funktionen $\mathcal{E}$.*

Beweis. f_3 ist eine elementare Funktion, denn es gilt:

$$\begin{aligned} f_3(0,y) &= f_2(y+1,y+1) = (y+2)*(y+2) = (y+2)^2 \\ f_3(1,y) &= f_3(0,f_3(0,y)) \quad = ((y+2)^2+2)^2 \\ f_3(2,y) &= f_3(1,f_3(1,y)) \quad = ((((y+2)^2+2)^2+2)^2+2)^2 \end{aligned}$$

Also lässt sich f_3 mithilfe der beschränkten Rekursion beschreiben:

$$\begin{aligned} g(y,0) &= y \\ g(y,x+1) &= (g(y,x)+2)^2 \\ g(y,x) &< (y+2)^{2^{2^{2x}}} \\ f_3(x,y) &= g(2^x,y) \end{aligned}$$

$p_1^2(x,y) = x + y \dot{-} y$ und $p_2^2(x,y) = x + y \dot{-} x$ sind ebenfalls elementare Funktionen. Da $\mathcal{E}$ abgeschlossen unter *Sub* und *lim.Rec* ist, folgt daraus:

$$\mathcal{E}^3 = \left[S(x), p_1^2, p_2^2, f_3(x,y)\,;\, Sub, lim.Rec\right] \subseteq \mathcal{E}$$

Auf der anderen Seite kann x^y in $\mathcal{E}^3$ definiert werden durch:

$$\begin{aligned} x+0 &= p_1^2(x,0) = x & \qquad x*0 &= p_2^2(x,0) = 0 \\ x+(y+1) &= (x+y)+1 & \qquad x*(y+1) &= (x*y)+x \\ x+y &\leq f_3(x,y) & \qquad x*y &\leq f_3(x,y) \end{aligned}$$

$$\begin{aligned} x^0 &= S(0) = 1 \\ x^{y+1} &= x^y * x \\ x^y &\leq f_3(y,x) \end{aligned}$$

Also gilt nach Satz 30: $\mathcal{E} = [S(x), x^y\,;\, Sub, lim.Rec] \subseteq \mathcal{E}^3$ und $\mathcal{E}^3 = \mathcal{E}$. □

Satz 51 (S. 32). *Alle Klassen $\mathcal{E}^n$ sind abgeschlossen unter der Operation des beschränkten Minimums.*

Beweis. Alle Klassen $\mathcal{E}^n$ enthalten die folgenden Funktionen:

$$p_1^3(x,y,z) = x = p_1^2(x, p_1^2(y,z))$$
$$p_2^3(x,y,z) = y = p_2^2(x, p_1^2(y,z))$$
$$p_3^3(x,y,z) = z = p_2^2(x, p_2^2(y,z))$$

$$0 \dot{-} 1 = p_1^2(0,0) \qquad x \dot{-} 0 = p_1^2(x,x)$$
$$(x+1) \dot{-} 1 = p_1^2(x, x \dot{-} 1) \qquad x \dot{-} (y+1) = p_3^3(x,y,(x \dot{-} y) \dot{-} 1)$$
$$x \dot{-} 1 \leq p_1^2(x,x) \qquad x \dot{-} y \leq p_1^2(x,y)$$

$$\sigma(x,0) = p_1^2(x,x) \qquad \tau(x,0) = x+1$$
$$\sigma(x,y+1) = p_1^2(0, p_1^3(x,y,\sigma(x,y))) \qquad \tau(x,y+1) = p_1^3(x,y,\tau(x,y))$$
$$\sigma(x,y) \leq p_1^2(x,y) \qquad \tau(x,y) \leq p_1^2(x+1,y)$$

Also ist $\sigma(x,y) = x * (1 \dot{-} y)$ und für alle $y > 0 : \tau(x,y) = x$. Dann enthalten alle Klassen für jede beliebige Funktion $f(\vec{u},x)$ auch die Funktion $g(\vec{u},x)$ mit:

$$g(\vec{u},0) = 1 \dot{-} f(\vec{u},0)$$
$$g(\vec{u},x+1) = \tau(g(\vec{u},x), f(\vec{u},x+1))$$
$$g(\vec{u},x) \leq p_2^2(f(\vec{u},0),x)$$

Dabei ist $g(\vec{u},x) = \sum_{i \leq x}(1 \dot{-} f(\vec{u},i))$ das Ergebnis der eingeschränkten Variante der beschränkten Addition. Folglich sind nach Korollar 17 alle $\mathcal{E}^n$ abgeschlossen unter der Operation des beschränkten Minimums. □

Satz 52 (S. 32). *Die Relationsmengen der Klassen $\mathcal{E}^n$ sind abgeschlossen unter den Operationen der Aussagenlogik, der Operation des beschränkten Existenzquantors und der Operation des beschränkten Allquantors.*

Beweis. Da alle $\mathcal{E}^n$ die Funktionen $S(x)$, $x \dot{-} y$ und $\sigma(x,y)$ enthalten und abgeschlossen sind unter Substitution, sind nach Korollar 13 alle Mengen der Relationen der $\mathcal{E}^n$ abgeschlossen unter den Operationen der Aussagenlogik.
Da die $\mathcal{E}^n$ außerdem abgeschlossen sind unter der Operation des beschränkten Minimums, sind die Relationsmengen nach Satz 22 abgeschlossen unter beiden Operationen der Beschränkten Quantoren. □

Satz 53 (S. 32).[6] *Für alle n gilt $\mathcal{E}^n \subseteq \mathcal{E}^{n+1}$.*

Beweis. Da sich die Definitionen der $\mathcal{E}^n$ nur um eine Initialfunktion unterscheiden, gilt für alle n, m: $\quad \mathcal{E}^m \subseteq \mathcal{E}^n \Leftrightarrow f_m \in \mathcal{E}^n$.
Für $\mathcal{E}^0$ ergibt sich für alle n: $f_0(x, y) = S(y) \in \mathcal{E}^n \Rightarrow \mathcal{E}^0 \subseteq \mathcal{E}^n$.
Die Definitionen aus dem Beweis von Satz 50 lassen sich in allen Klassen mit ausreichend großen Schrankenfunktionen wiederholen. Daher gilt:

$$\begin{aligned} f_1(x, y) = x + y \leq f_2(x, y) \qquad & \Rightarrow \mathcal{E}^1 \subseteq \mathcal{E}^2 \\ f_2(x, y) = (x + 1) * (y + 1) \in \mathcal{E}^3 \qquad & \Rightarrow \mathcal{E}^2 \subseteq \mathcal{E}^3 \\ \forall n \geq 3 : \quad x^y \leq f_3(x, y) \leq f_n(x, y) \qquad & \Rightarrow \mathcal{E}^3 \subseteq \mathcal{E}^n \end{aligned}$$

Da die $f_n(x, y)$ nach Satz 47 in jedem Punkt eine aufsteigende Kette bilden, gilt für alle $n > 3$:

$$\mathcal{E}^0 \subseteq \mathcal{E}^1 \subseteq \mathcal{E}^2 \subseteq \mathcal{E}^3 \subseteq \mathcal{E}^n$$

Nun treffen wir die Induktionsannahme, dass die Klasse $\mathcal{E}^n$ ein f_m für $2 \leq m < n$ enthält. Für die Funktion f_{m+1} gilt dann:

$$\begin{aligned} f_{m+1}(0, j) &= f_m(j + 1, j + 1) \\ f_{m+1}(i + 1, j) &= f_{m+1}(i, f_{m+1}(i, j)) \\ f_{m+1}(i, j) &\leq f_n(i, j) \end{aligned}$$

Diese Bedingungen folgen zwar nicht der Definition von beschränkter Rekursion, können aber mit einer ähnlichen Methode, wie in Satz 29 durch das beschränkte Minimum umgesetzt werden. In diesem Fall ist $f_{m+1}(x, y) = z$ äquivalent dazu, dass es eine Matrix mit Einträgen $a_{i,j}$ gibt, für die gilt:

$$a_{x,y} = z \tag{a}$$

$$\forall i, j : a_{i+1,j} \neq 0 \rightarrow \exists k : a_{i+1,j} = a_{i,k} \wedge k \neq 0 \wedge k = a_{i,j} \tag{b}$$

$$\forall j : a_{0,j} \neq 0 \rightarrow a_{0,j} = f_m(j + 1, j + 1) \tag{c}$$

Die Matrix enthält also alle Werte von $f_{m+1}(i, j) = a_{i,j}$, die für die Berechnung von $f_{m+1}(x, y) = z$ benötigt werden und 0 an allen anderen Stellen. Da für alle $m > 1$ nach Satz 44 $f_m(i, j) > j \geq 0$ gilt, ist 0 als Repräsentation der leeren Einträge geeignet.[7]

[6] Den ersten Teil von Grzegorczyks Beweis habe ich ausgelagert in Satz 47.
[7] Diesen Fakt hat Grzegorczyk nicht beachtet, weshalb seine Exponenten überall um 1 größer sind.

Bedingung (a) gibt das Ziel vor: $f_{m+1}(x, y) = z$.

Falls $f_{m+1}(i + 1, j)$ für die Berechnung benötigt wird, also $a_{i+1,j} \neq 0$ ist, dann muss es nach Bedingung (b) ein $k \neq 0$ geben, sodass

$$\begin{aligned} f_{m+1}(i + 1, j) &= f_{m+1}(i, k) \\ k &= f_{m+1}(i, j) \\ \Rightarrow f_{m+1}(i + 1, j) &= f_{m+1}(i, f_{m+1}(i, j)) \end{aligned}$$

Zuletzt deckt Bedingung (c) den Fall $f_{m+1}(0, j) = f_m(j + 1, j + 1)$ ab.

Die Matrix soll nun in einer Zahl kodiert werden. Dazu benötigen wir zunächst eine injektive Paarfunktion $P : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, die jedem Index (i, j) eine eindeutige Zahl zuordnet. Eine Möglichkeit dafür ist zum Beispiel $P(x, y) = 2^x * 3^y$. Diese Sequenz von Indizes lässt sich dann mithilfe der Primzahlen kodieren. Daraus ergibt sich die folgende Relation[8]:

$$R(x, y, z) = \bigvee_{a \leq m(x,y)} \bigwedge_{j \leq a} \left(\begin{array}{c} \operatorname{prex}(a, P(x, y)) = z \wedge \\ \bigwedge\limits_{i \leq a} \left(\begin{array}{l} \operatorname{prex}(a, P(i + 1, j)) \neq 0 \\ \rightarrow \bigvee\limits_{k \leq a} \left(\begin{array}{r} \operatorname{prex}(a, P(i + 1, j)) = \operatorname{prex}(a, P(i, k)) \\ \wedge\, k \neq 0 \wedge k = \operatorname{prex}(a, P(i, j)) \end{array} \right) \end{array} \right) \wedge \\ \Big(\operatorname{prex}(a, P(0, j)) \neq 0 \rightarrow \operatorname{prex}(a, P(0, j)) = f_m(j + 1, j + 1) \Big) \end{array} \right)$$

Diese Gleichung beruht darauf, dass $P(i, j)$ in beiden Argumenten monoton wachsend ist. Die Matrix hat höchstens $(x+1) \times f_n(x, y)$ Einträge und alle Einträge sind maximal so groß wie $f_n(x, y)$. Daher eignet sich als Schrankenfunktion[9]:

$$m(x, y) = \operatorname{prim}(P(x, f_n(x, y)))^{(x+1)*f_n(x,y)^2}$$

Da $\mathcal{E}^n$ alle elementaren Funktionen, sowie $f_n(x, y)$ und $f_m(x, y)$ enthält, ist R eine Relation von $\mathcal{E}^n$. Schließlich lässt sich $f_{m+1}(x, y)$ definieren durch:

$$f_{m+1}(x, y) = \mu(z \leq f_n(x, y))\,[R(x, y, z)]$$

Nach diesem Induktionsschritt gilt also für alle $m < n : f_m(x, y) \in \mathcal{E}^n$. □

[8] Aus Gründen dar Darstellbarkeit habe ich die Definition etwas umgeformt und den Existenzquantor über k eingefügt.

[9] Die Schranke von Grzegorczyk scheint mehr Ansprüche an $P(i, j)$ zu haben. Um das zu vermeiden, habe ich sie etwas verändert.

Lemma 54 (S. 35). *Ist $g(x) \in \mathcal{E}_1^0$ eine Funktion der Ordnung k, dann gilt:*

$$g(x) \leq x + 2^k$$

Beweis. **Induktion** über k:

Die einstellige Initialfunktion von $\mathcal{E}^0$ erfüllt die Bedingung:

$$S(x) = x + 2^0 \leq x + 2^0$$

Alle Initialfunktionen erfüllen die Bedingung zumindest bezüglich eines Arguments:

$$\begin{aligned} p_1^2(x, y) = x &\leq x + 2^0 \\ p_2^2(x, y) = y &\leq y + 2^0 \\ f_0(x, y) = S(y) &\leq y + 2^0 \end{aligned}$$

Äquivalent zu dieser Aussage wäre $f(x_1, \dots, x_n) \leq \max\{x_1, \dots, x_n\}$.

Wenn alle Funktionen g, h mit Ordnung von höchstens k die Bedingung für alle Parameter zumindest bezüglich eines Arguments erfüllen, dann folgt daraus:

a) Wenn F durch Substitution von h in g entsteht, gilt:

$$\begin{aligned} &F(x_1, \dots, x_{l-1}, x_{l+1}, \dots, x_n, y_1, \dots, y_m) \\ &\qquad = g(x_1, \dots, x_{l-1}, h(y_1, \dots, y_m), x_{l+1}, \dots, x_n) \end{aligned}$$

Für alle $x_1, \dots, x_n$ gibt es ein i mit $g(x_1, \dots, x_n) \leq x_i$.
Für alle $y_1, \dots, y_m$ gibt es ein j mit $h(y_1, \dots, y_m) \leq y_j$.

Falls x_i die substituierte Variable x_l ist, gilt:

$$F(...) \leq h(y_1, \dots, y_m) + 2^k \leq y_j + 2^k + 2^k \qquad = y_j + 2^{k+1}$$

Sonst: $$F(...) \leq x_i + 2^k \qquad < x_i + 2^{k+1}$$

Also gilt für F zumindest bezüglich eines Arguments z_r:

$$F(z_1, \dots, z_{n+m-1}) \leq z_r + 2^{k+1}$$

b) Wenn F durch Identifikation von Variablen aus g entsteht, ist leicht zu sehen, dass die Bedingung von g erhalten bleibt.

c) Wenn F aus g durch Substitution von 0 entsteht, gilt entweder

$$F(x_1, \dots, x_{l-1}, x_{l+1}, \dots, x_n) \leq 0 + 2^k$$

oder die Bedingung bleibt ebenfalls erhalten.

d) Wenn F durch beschränkte Rekursion mit g als Schrankenfunktion entsteht, ist $F(\vec{u}) \leq g(\vec{u})$ und die Bedingung bleibt erhalten.

In jedem Fall gilt demnach für alle Funktionen F der Ordung $k+1$ die Bedingung $F(z_1, \dots, z_s) \leq z_r + 2^{k+1}$ für zu mindest ein r.

Nach dem Induktionsschritt gilt das also für alle Funktionen aus $\mathcal{E}^0$. Für alle einstelligen Funktionen $g(x) \in \mathcal{E}_1$ mit Ordnung k bedeutet das:

$$g(x) \leq x + 2^k$$

□

Lemma 55 (S. 35). *Ist $h(x) \in \mathcal{E}_1^1$ eine Funktion der Ordnung k, dann gilt:*

$$h(x) \leq (x+1) * 2^{2^{2k}}$$

Beweis. Der Beweis ist beinahe der gleiche wie bei Lemma 54 mit einer zusätzlichen Initialfunktion $f_1(x, y) = x + y$, für die gilt:

$$\begin{aligned} &\text{Für } x > y \text{ gilt} \quad x + y < x + x < (x+1) * 2^{2^0}. \\ &\text{Für } x \leq y \text{ gilt} \quad x + y \leq y + y < (y+1) * 2^{2^0}. \end{aligned}$$

Für die anderen Initialfunktionen folgt die gleiche Bedingung bezüglich mindestens eines Arguments durch:

$$x + 2^0 = (x+1) * 2^0 < (x+1) * 2^{2^0}$$

Im Induktionsschritt, wenn F aus g und h durch Substitution entsteht, gilt dann, falls g die Bedingung bezüglich der substituierten Variable erfüllt:

$$\begin{aligned} F(...) \leq (h(y_1, \dots, y_m) + 1) * 2^{2^{2k}} &\leq ((y_j + 1) * 2^{2^{2k}} + 1) * 2^{2^{2k}} \\ &= (y_j + 1) * 2^{2^{2k+1}} + 2^{2^{2k}} \leq (y_j + 1) * 2^{2^{2k+2}} \end{aligned}$$

In allen weiteren Fällen bleibt die Bedingung erhalten. □

Satz 56 (S. 35). *Für alle n wächst die Funktion $f_{n+1}(x,x)$ schneller als jede Funktion in $\mathcal{E}_1^n$.*

Beweis. Für diesen Beweis benötigen wir zunächst die Funktionsklassen

$$\mathcal{W}^n = \left[I(x,y)\,, K(z)\,, L(z)\,, x+y, x^2, f_n(x,y)\,;\; Sub\right].$$

Nach Satz 35[10]gilt für $\mathcal{W}_1^n$:

$$\mathcal{W}_1^n = \left[\begin{matrix} K(x)\,, L(x)\,, \\ I(x,x)\,, I(x,0)\,, I(0,x) \end{matrix}, K(x)+L(x)\,, x^2, f_n(K(x)\,, L(x))\,;\; \circ, \mathbf{O}_{ILK}\right]$$

Der Rest des Beweises erfolgt über ein paar Zwischenschritte.

Lemma 57 (S. 35).[10] *Für alle $g(x) \in \mathcal{W}_1^n$ mit Ordnung k gilt:*

$$g(x) < f_{n+1}(k+1, x)$$

Beweis. **Induktion** über die Ordnung von Funktionen in $\mathcal{W}_1^n$:

Für die Initialfunktionen von $\mathcal{W}_1^n$ gilt:

$$\begin{gathered} x^2 < (x+2)^2 = f_3(0,x) \leq f_{n+1}(0,x) \\ I(x,0)\,, I(0,x) \leq I(x,x) = (x+x)^2 + x < ((x+2)^2+2)^2 = f_3(1,x) \\ K(x)\,, L(x) \leq x < f_{n+1}(0,x) \\ f_n(K(x)\,, L(x)) < f_n(x+1, x+1) = f_{n+1}(0,x) \end{gathered}$$

Wenn $g, h \in \mathcal{W}_1^n$ Funktionen mit Ordnung kleiner als k sind und $g(x), h(x) < f_{n+1}(k,x)$ gilt, dann folgt daraus:

$$g(h(x)) < f_{n+1}(k, h(x)) < f_{n+1}(k, f_{n+1}(k,x)) = f_{n+1}(k+1, x)$$

$$\begin{aligned} I(g(K(x)), L(x)) &< I(f_{n+1}(k,x), f_{n+1}(k,x)) < f_3(1, f_{n+1}(k,x)) \\ I(K(x)\,, g(L(x))) &< I(f_{n+1}(k,x), f_{n+1}(k,x)) \\ f_3(1, f_{n+1}(k,x)) &\leq f_{n+1}(k, f_{n+1}(k,x)) \qquad = f_{n+1}(k+1,x) \end{aligned}$$

□

[10] Grzegorczyk hat bei der Anwendung von Satz 35 ein etwas anderes Ergebnis erhalten. Da das nicht direkt nachvollziehbar ist, habe ich mich an die dort gegebene Definition gehalten. Dadurch ändert sich in Lemma 57 der Beweis und die Ordnung um 1, aber die Folgerung bleibt die gleiche.

Daraus folgt direkt $g(x) < f_{n+1}(x,x)$ für alle $x > k+1$ und somit wächst $f_{n+1}(x,x)$ schneller als jede Funktion in $\mathcal{W}_1^n$. Außerdem benötigen wir:

Lemma 58 (S. 35). *Jede Funktion in $\mathcal{E}^n$ wird von einer Funktion aus $\mathcal{W}^n$ dominiert.*

Beweis. Da alle Initialfunktionen von $\mathcal{E}^n$ von einer monoton wachsenden Funktion aus $\mathcal{W}^n$ dominiert werden, beweist Satz 42 dieses Lemma:

$$\begin{aligned} S(x) = f_0(0,x) &\leq f_n(0,x) \in \mathcal{W}^n \\ p_1^2(x,y) = x &\leq f_n(x,y) \in \mathcal{W}^n \\ p_2^2(x,y) = y &\leq f_n(x,y) \in \mathcal{W}^n \\ f_n(x,y) &\leq f_n(x,y) \in \mathcal{W}^n \end{aligned}$$ □

Mit Hilfe der zwei Lemmata können wir nun Satz 43 anwenden.
Für alle $n > 1$ wächst $f_{n+1}(x,x)$ schneller als jede Funktion in $\mathcal{W}_1^n$ und damit auch schneller als jede Funktion in $\mathcal{E}_1^n$. Für die verbleibenden zwei Fälle verwenden wir Lemma 54 und Lemma 55. Für alle $g(x) \in \mathcal{E}_1^0$ und $h(x) \in \mathcal{E}_1^1$ mit Ordnung k gilt:

$$\begin{aligned} g(x) \leq x + 2^k \quad &\Rightarrow \text{ Für alle } x > 2^k \text{ gilt } g(x) < x + x = f_1(x,x). \\ h(x) \leq (x+1) * 2^{2^{2k}} &\Rightarrow \text{ Für alle } x > 2^{2^{2k}} \text{ gilt } h(x) < (x+1)^2 = f_2(x,x). \end{aligned}$$

Folglich gilt der Satz für alle $n \in \mathbb{N}$. □

Korollar 59 (S. 36). *Eine direkte Folgerung aus diesem Satz ist:*

$$\textit{Für alle } n \in \mathbb{N} \textit{ ist} \quad f_{n+1}(x,x) \in \mathcal{E}^{n+1}, \quad \textit{aber} \quad f_{n+1}(x,x) \notin \mathcal{E}^n.$$

Die Grzegorczyk Hierarchie bildet also eine streng monoton wachsende Folge von Funktionsklassen.

$$\textit{Für alle } n \in \mathbb{N} \textit{ gilt } \mathcal{E}^n \subsetneq \mathcal{E}^{n+1}.$$

2.4.3 Primitive Rekursion

Definition 60. Eine Funktionsklasse heißt abgeschlossen unter der **Operation der (primitiven) Rekursion**, wenn sie für alle Funktionen $f(\vec{u})$ und $g(\vec{u},x,y)$ auch die Funktion $F(\vec{u},x)$ enthält, sodass für alle $\vec{u}$ und x gilt:

$$\begin{aligned} F(\vec{u},0) &= f(\vec{u}) \\ F(\vec{u},x+1) &= g(\vec{u},x,F(\vec{u},x)) \end{aligned}$$

Die Funktion wird genau so definiert wie bei beschränkter Rekursion, nur ohne eine Schranke anzugeben. Die primitive Rekursion wird abgekürzt durch *Rec*.

Definition 61. Die Klasse der primitiv-rekursiven Funktionen ist:

$$\mathcal{R} = \left[S(x), p_1^2, p_2^2 \,;\, Sub, Rec\right]$$

Lemma 62 (S. 39). *Alle $f_n(x, y)$ sind primitiv-rekursive Funktionen.*

Beweis. Wir haben bereits rekursive Definitionen von $x + y$ und $x * y$ verwendet. Durch Substitution entstehen daraus die Funktionen f_0, f_1 und f_2. Alle weiteren f_n entstehen nach einem Schema, das Rózsa Péter als eingeschachtelte Rekursion bezeichnet. In ihrem Artikel „Über den Zusammenhang der verschiedenen Begriffe der rekursiven Funktion“ von 1935 [3, § 2] hat sie bewiesen, dass die primitiv-rekursiven Funktionen unter dieser Operation abgeschlossen sind. Durch **Induktion** über n lässt sich dann zeigen, dass alle f_n primitiv-rekursive Funktionen sind. □

Lemma 63 ([5, S. 940]). *Die Klasse der einstelligen primitiv-rekursiven Funktionen ist:*

$$\mathcal{R}_1 = [S(x), K(x) \;;\; +, \circ, \circ^x]$$

Dabei ist $\circ^x$ die x-fache Komposition einer Funktion mit Argument 0.

$$\textit{Aus } f(x) \textit{ entsteht die Funktion } g(x) = f^x(0) = f \circ ... \circ f(0).$$

Dieses Resultat stammt aus dem Paper „Primitive Recursive Functions“ von Raphael M. Robinson (1948) [5] und wird hier nicht bewiesen.

Lemma 64 (S. 39). *Ist $f \in \mathcal{R}_1$ eine Funktion der Ordnung k, dann ist $f \in \mathcal{E}^{k+3}$.*

Beweis. **Induktion** über die Ordnung von f in $\mathcal{R}_1$:

Die Initialfunktionen $S(x)$ und $K(x)$ von $\mathcal{R}_1$ liegen in $\mathcal{E}^3$.

Wenn g und h der Ordnung k in $\mathcal{E}^{k+3}$ liegen, dann folgt daraus

$$g(x) + h(x) \in \mathcal{E}^{k+3} \quad \text{und} \quad g(h(x)) \in \mathcal{E}^{k+3},$$

denn $\mathcal{E}^{k+3}$ enthält $x + y$ und ist abgeschlossen unter Substitution.
Außerdem wissen wir aus Lemma 58, dass jede Funktion aus $\mathcal{E}^{k+3}$ von einer Funktion aus $\mathcal{W}^{k+3}$ dominiert wird. Sei nun $f' \in \mathcal{W}^{k+3}$ eine Funktion der Ordnung l, die f dominiert. Dann gilt nach Lemma 57:

$$f(x) \leq f'(x) < f_{k+4}(l + 1, x)$$

Induktion über y:

Für $y = 0$ ist $f^0(0) = id(0) = 0 < f_{k+4}(l+1, 0)$.

Wenn $f^y(0) < f_{k+4}(l+y+1, 0)$ ist, dann folgt daraus:

$$\begin{aligned} f^{y+1}(0) = f(f^y(0)) &< f_{k+4}(l+y+1, f^y(0)) < f_{k+4}(l+1, f_{k+4}(l+y+1, 0)) \\ &\leq f_{k+4}(l+y+1, f_{k+4}(l+y+1, 0)) = f_{k+4}(l+y+2, 0) \end{aligned}$$

Folglich gilt also für alle y: $f^y(0) < f_{k+4}(l+y+1, 0)$. Damit lässt sich nun $h(x) = f^x(0)$ definieren durch:

$$\begin{aligned} h(0) &= 0 \\ h(x+1) &= f(h(x)) \\ h(x) &< f_{k+4}(l+x+1, 0) \end{aligned}$$

Es gilt also $f^x(0) \in \mathcal{E}^{k+4}$ und das schließt den Induktionsschritt ab. □

Satz 65 (S. 39). *Die Vereinigung aller Klassen $\mathcal{E}^n$ ist die Klasse der primitiv-rekursiven Funktionen.*

Beweis. Alle Initialfunktionen von $\mathcal{E}^0$ sind auch Initialfunktionen von $\mathcal{R}$. Mit Lemma 62 liegen die Initialfunktionen aller $\mathcal{E}^n$ in $\mathcal{R}$. Alle Funktionen, die durch beschränkte Rekursion entstehen, entstehen auch durch primitive Rekursion. Folglich gilt für alle n:

$$\mathcal{E}^n \subsetneq \mathcal{R}.$$

Aus Lemma 64 folgt direkt für alle $f(x_1, \dots, x_n) \in \mathcal{R}$:
Wenn $f'(x) = f(C_1^n(x), \dots, C_n^n(x)) \in \mathcal{R}_1$ Ordnung k hat, dann gilt:

$$f'(x) \in \mathcal{E}^{k+3} \quad \Rightarrow \quad f(x_1, \dots, x_n) = f'(T(x_1, \dots, x_n)) \in \mathcal{E}^{k+3} \subsetneq \bigcup_{n=0}^{\infty} \mathcal{E}^n$$

Insgesamt gilt also:

$$\bigcup_{n=0}^{\infty} \mathcal{E}^n \subseteq \mathcal{R} \text{ und } \mathcal{R} \subseteq \bigcup_{n=0}^{\infty} \mathcal{E}^n \quad \Leftrightarrow \quad \bigcup_{n=0}^{\infty} \mathcal{E}^n = \mathcal{R}$$ □

Die Grzegorczyk Hierarchie kategorisiert also alle primitiv-rekursiven Funktionen anhand ihres Wachstums. An dieser Stelle verbleibt noch die Frage, was das mit der Berechenbarkeit und Komplexität der Funktionen zu tun hat.

3 Die Intrinsische Komplexität von Funktionen

Das letzte Kapitel behandelt das Paper „The Intrinsic Computational Difficulty of Functions“ von Alan Cobham aus dem Jahr 1965. Darin baut er auf einigen Resultaten von Robert W. Ritchie auf, die unter dem Titel „Classes of predictably computable Functions“ zwei Jahre zuvor veröffentlicht wurden. Diese beiden Paper zusammen bilden einen guten Überblick des Zusammenhangs zwischen den Klassen der Grzegorczyk Hierarchie und der Komplexität von Turingmaschinen.
Als Motivation für diese Untersuchung hat Cobham die Frage aufgeworfen, ob Multiplikation schwieriger ist als Addition. Jeder, der sich schon mal mit den beiden Verfahren beschäftigt hat, könnte diese Frage mit „Ja“ beantworten. Doch ein formaler Beweis dieser Aussage scheint einige tieferliegende Probleme aufzuwerfen.

3.1 Grzegorczyk und Turing

Ritchie hat eine Hierarchie von Klassen F_i definiert, in der die Funktionen durch den Speicherbedarf einer Turingmaschine beschränkt sind. Alle F_i enthalten die Klasse $\mathcal{E}^2$ und sind enthalten in $\mathcal{E}^3$. Verwendet werden dabei nur Turingmaschinen mit einem einzigen Band für Ein- und Ausgabe.

Definition 66. Eine Turingmaschine berechnet eine Funktion $f : \mathbb{N}^n \to \mathbb{N}$, wenn für jede Eingabe $x_1\beta...\beta x_n$ nach der Berechnung der Wert $f(x_1, ..., x_n)$ auf dem Band verbleibt. Dabei werden die Zahlen in einer bestimmten polyadischen Darstellung auf das Band geschrieben und durch ein gesondertes Symbol β voneinander getrennt.

Wie im folgenden Abschnitt deutlich wird, ist es unerheblich, welche Basis für die Zahlendarstellung gewählt wird. Des Weiteren definieren wir:

Definition 67. $l(x_1, ..., x_n)$ ist die Anzahl an Bandzellen, die benötigt werden, um die Zahlen $x_1, ..., x_n$ zu kodieren. Die Trennsymbole werden dabei mitgezählt.

Das bedeutet, mit b-adischer Zahlendarstellung gilt:

$$l(x_1, ..., x_n) = (n-1) + \sum_{i=1}^{n} \lceil \log_b(x_i + 1) \rceil$$

Falls die 0 nicht mit dem leeren Wort sondern mit 0 kodiert werden soll, gilt:

$$l(x_1, \dots, x_n) = (n-1) + \sum_{i=1}^{n} \lceil \log_b(x_i + 1) \rceil + (1 \dot{-} x_i)$$

Definition 68 ([vgl. 1, S. 25]). Für eine 1-Band-Turingmaschine Z, die eine totale Funktion $f : \mathbb{N}^n \to \mathbb{N}$ berechnet, sei $\sigma_Z(x_1, \dots, x_n)$ die Anzahl an Schritten und $\tau_Z(x_1, \dots, x_n)$ die Anzahl an gelesenen Bandzellen, die sie dabei auf der Eingabe $x_1, \dots, x_n$ benötigt.

Wenn man mehrere Turingmaschinen definiert hat, dann lässt sich daraus eine weitere Maschine konstruieren, die die einzelnen Programme nacheinander ausführt. Wenn man die Maschinen geschickt verknüpft, dann lassen sich durch diesen Prozess verschiedene Operationen auf den berechneten Funktionen realisieren.

Satz 69 ([vgl. 4, S. 141]).[1] *Seien $f(\vec{x})$ und $g(\vec{y})$ totale Funktionen, die von den Turingmaschinen F und G berechnet werden. Dann gibt es für jede Funktion h, eine Turingmaschine, die sie berechnet, mit folgenden Eigenschaften:*

1. *Wenn $h(\vec{x}_1, \vec{x}_2, \vec{y})$ durch Substitution von g in f entsteht, gilt:*

$$\tau_H(\vec{x}_1, \vec{x}_2, \vec{y}) = l(\vec{x}_1, \vec{x}_2, \vec{y}) + \max\left\{\tau_F(\vec{x}_1, g(\vec{y}), \vec{x}_2),\ l(\vec{x}_1, \vec{x}_2) + \tau_G(\vec{y})\right\}$$

2. *Wenn $h(\vec{x}_1, \vec{x}_2, \vec{x}_3, y)$ durch Identifikation von Variablen aus f entsteht, gilt:*

$$\tau_H(\vec{x}_1, \vec{x}_2, \vec{x}_3, y) = l(\vec{x}_1, \vec{x}_2, \vec{x}_3, y) + \tau_F(\vec{x}_1, y, \vec{x}_2, y, \vec{x}_3)$$

3. *Wenn $h(\vec{x}_1, \vec{x}_2)$ durch Substitution von 0 in f entsteht, gilt:*

$$\tau_H(\vec{x}_1, \vec{x}_2) = l(\vec{x}_1, \vec{x}_2) + \tau_F(\vec{x}_1, 0, \vec{x}_2)$$

4. *Wenn $h(\vec{u}, y)$ durch primitive Rekursion aus f und g entsteht, gilt:*

$$\tau_H(\vec{u}, y) = 2 * \left(l(\vec{u}, y) + \max\left\{\tau_F(\vec{u}),\ \max_{z \le y}\left\{\tau_G(\vec{u}, z, h(\vec{u}, z))\right\}\right\}\right)$$

Beweis. Die vier Beweise geschehen recht einfach durch eine allgemeine Konstruktion einer Turingmaschine, die die jeweilige Operation umsetzt. Diese Maschinen werden hier lediglich konzeptuell beschrieben. Eine Maschine, die g in f einsetzt, könnte zunächst die für g benötigten Argumente in der richtigen Reihenfolge hinter die Eingabe kopieren

[1] Ritchie hat hier statt 2 und 3 das ähnliche Konzept der expliziten Transformation verwendet. Diese beiden Varianten sind allerdings nur äquivalent, wenn $S(x)$, p_1^2 und p_2^2 zur Verfügung stehen.

und nur darauf die Berechnung von g ausführen. Dann muss das Ergebnis zusammen mit den restlichen Argumenten nur noch an die richtige Stelle verschoben werden, damit f berechnet werden kann. Es ist leicht zu sehen, dass dies innerhalb des oben beschriebenen Speichers möglich ist.
Für 2 und 3 müssen die Argumente nur vor der Berechnung von f an die richtige Stelle kopiert bzw. durch 0 ersetzt werden. Für die Umsetzung der primitiven Rekursion muss am Anfang einmal f und dann mehrmals g berechnet werden. Dabei müssen außerdem die Anzahl der Durchläufe gezählt und die richtigen Argumente für g berechnet werden.□

Ritchie hat die Klassen F_i auch algebraisch charakterisiert. Dabei verwendete er eine Arithmetisierung von Turingmaschinen in $\mathcal{E}^2$ [vgl. 4, S. 167]. Das heißt, er definierte die Funktionen $\text{Init}_n, \text{Yield}, \text{Decode}$ in $\mathcal{E}^2$, sowie eine Relation Fin von $\mathcal{E}^2$. Diese beschreiben die Berechnung einer Turingmaschine wie folgt:

- $\text{Init}_n(x_1, \dots, x_n)$ ist die Gödelnummer der Konfiguration einer (beliebigen) Turingmaschine im Startzustand, die die Argumente $x_1, \dots, x_n$ als Bandinhalt hat.
- $\text{Yield}(x, z)$ führt einen Berechnungsschritt der Turingmaschine mit Gödelnummer z aus. Sie berechnet also aus der Gödelnummer x einer Konfiguration die Nummer der nächsten Konfiguration.
- $\text{Decode}(x)$ ist die Zahl, die von dem Bandinhalt der Konfiguration mit Gödelnummer x repräsentiert wird, sofern das Band eine korrekt kodierte Zahl enthält und 0 sonst.
- $\text{Fin}(x)$ gibt an, ob sich die Konfiguration mit Gödelnummer x in einem Endzustand befindet.

Durch primitive Rekursion lässt lassen sich damit Berechnungen von Turingmaschinen durchführen. Für jede korrekte Gödelisierung z gibt es:

$$\begin{aligned} H_{n,z}(x_1, \dots, x_n, 0) &= \text{Init}_n(x_1, \dots, x_n) \\ H_{n,z}(x_1, \dots, x_n, y+1) &= \text{Yield}(H_{n,z}(x_1, \dots, x_n, y), z) \end{aligned}$$

Wenn eine totale Funktion $f(x_1, \dots, x_n)$ von einer Turingmaschine Z mit Gödelisierung z berechnet wird, dann gilt:

$$f(\vec{x}) = \text{Decode}\left(H_{n,z}\left(\vec{x}, \mu(y \leq \sigma_Z(\vec{x}))\left[\text{Fin}\left(H_{n,z}(\vec{x}, y)\right)\right]\right)\right)$$

Der Wert von f ist also die Zahl auf dem Band, nach der kleinst möglichen Anzahl von Schritten, die zu einem Endzustand führen. Die Anzahl der benötigten Schritte ist dabei genau $\sigma_Z(\vec{x})$. Daraus folgt:

Satz 70. *Wenn Z die Funktion f berechnet und σ_Z primitiv-rekursiv ist, dann ist auch f primitiv-rekursiv.*

Beweis. $\mathcal{R}$ ist abgeschlossen unter Substitution und der Operation des beschränkten Minimums und Decode, $H_{n,z}$, sowie die Funktion zu Fin sind primitiv-rekursiv. □

An dieser Stelle hat Cobham festgestellt, dass sich die primitiv-rekursiven Funktionen so noch weiter klassifizieren lassen:

Satz 71 ([vgl. 1, S. 25]). *Für alle $n > 2$ sind die folgenden Aussagen äquivalent:*

1. $f \in \mathcal{E}^n$
2. *Es gibt eine Turingmaschine Z, die f berechnet, sodass $\sigma_Z \in \mathcal{E}^n$.*
3. *Es gibt eine Turingmaschine Z, die f berechnet, und $g \in \mathcal{E}^n$, sodass σ_Z dominiert wird von g, also für alle $\vec{x}$ gilt $\sigma_Z(\vec{x}) \leq g(\vec{x})$.*
4. *Es gibt eine Turingmaschine Z, die f berechnet, sodass $\tau_Z \in \mathcal{E}^n$.*
5. *Es gibt eine Turingmaschine Z, die f berechnet, und $g \in \mathcal{E}^n$, sodass τ_Z dominiert wird von g, also für alle $\vec{x}$ gilt $\tau_Z(\vec{x}) \leq g(\vec{x})$.*

Beweis. **(3 ⇒ 1)** Für jede Turingmaschine ist $\tau_Z(\vec{x}) \leq \sigma_Z(\vec{x}) \leq g(\vec{x})$. Wenn m die größte Gödelnummer aller Zustände und Bandsymbole ist, dann ist die Gödelnummer jeder Konfiguration in der Berechnung mit Sicherheit kleiner als

$$(\text{prim}(\sigma_Z(\vec{x}) + 1)^m)^{\sigma_Z(\vec{x})+1} \leq (\text{prim}(g(\vec{x}) + 1)^m)^{g(\vec{x})+1} = q'(\vec{x})$$

[vgl. 4, S. 148]. Auf die Details der Gödelisierung werde ich nicht weiter eingehen. Wenn g in $\mathcal{E}^n$ liegt, dann auch q', denn $\text{prim}(k)$ und x^y sind elementar. Diese größte Gödelnummer ist eine obere Schranke für $H_{n,z}$, also $H_{n,z}(\vec{x}, y) \leq q'(\vec{x})$. Damit entsteht $H_{n,z}$ durch beschränkte Rekursion und liegt ebenfalls in $\mathcal{E}^n$. Da $\mathcal{E}^n$ abgeschlossen ist unter der Operation des beschränkten Minimums, gilt:

$$f(\vec{x}) = \text{Decode}\left(H_{n,z}\left(\vec{x}, \mu(y \leq g(\vec{x}))\left[\text{Fin}\Big(H_{n,z}(\vec{x}, y)\Big)\right]\right)\right) \in \mathcal{E}^n$$

(3 ⇒ 2) Wie oben ist $H_{n,z}$ beschränkt durch q'. Daraus folgt:

$$\sigma_Z(\vec{x}) = \mu(y \leq g(\vec{x}))\left[\text{Fin}\Big(H_{n,z}(\vec{x}, y)\Big)\right] \in \mathcal{E}^n$$

(2 ⇒ 3) Es gilt $\sigma_Z(\vec{x}) \leq \sigma_Z(\vec{x}) \in \mathcal{E}^n$.

(5 ⇒ 3) Da Turingmaschinen für totale Funktionen bei richtig kodierter Eingabe immer halten, lässt sich aus dem Speicherbedarf eine Laufzeitschranke konstruieren. Die Laufzeit kann nicht größer sein als die Anzahl der möglichen Konfigurationen $h(\vec{x})$. Da der Speicher durch g beschränkt ist, gilt:

$$h(\vec{x}) \leq s * \tau_Z(\vec{x}) * a^{\tau_Z(\vec{x})} \leq s * g(\vec{x}) * a^{g(\vec{x})} = h'(\vec{x})$$

Dabei ist s die Anzahl der Zustände und a die Anzahl der Bandsymbole. Wenn $g \in \mathcal{E}^n$ liegt, dann auch h' und für alle $\vec{x}$ gilt: $\sigma_Z(\vec{x}) \leq h'(\vec{x}) \in \mathcal{E}^n$.

(4 ⇒ 5) Es gilt $\tau_Z(\vec{x}) \leq \tau_Z(\vec{x}) \in \mathcal{E}^n$.

(1 ⇒ 4) Für den letzten Teil benötigen wir Satz 69. Sei $\mathcal{E}^n_\tau$ die Klasse der Funktionen, die sich mit $\tau_Z \in \mathcal{E}^n$ berechnen lassen. Mittels **Induktion** beweisen wir $\mathcal{E}^n \subseteq \mathcal{E}^n_\tau$:

Für alle m ist $l(x_1, \ldots, x_m) \in \mathcal{E}^3$, denn mit einer b-adischen Kodierung gilt:

$$l(x_1) = \mu(y \leq x)\,[x < b^y] + (1 \dot{-} x)$$
$$l(x_1, \ldots, x_{m+1}) = l(x_1, \ldots, x_m) + 1 + l(x_{m+1}) \in \mathcal{E}^3$$

Außerdem lässt sich das Maximum zweier Zahlen berechnen durch:

$$\max(x, y) = (1 \dot{-} (y \dot{-} x)) * x + (1 \dot{-} (x + 1 \dot{-} y)) * y \in \mathcal{E}^3$$

Induktion über die Ordnung in $\mathcal{E}^3 = [S(x), x^y\,;\, \mathit{Sub}, \mathit{lim.Rec}]$:

> Die Nachfolgerfunktion $S(x)$ lässt sich mit $\tau_Z(x) = l(x) + 1 \in \mathcal{E}^3$ berechnen.
> x^y lässt sich mit $\tau_Z(x, y) = l(x, y) + l(x) * y + 1 \in \mathcal{E}^3$ berechnen.
>
> Wenn h durch Substitution aus Funktionen f und g von niedrigerer Ordnung entsteht, dann ist $h \in \mathcal{E}^3_\tau$, da τ_H in allen drei Fällen durch Substitution aus τ_F, τ_G, $l(\vec{x})$, $x + y$, $\max(x, y)$ und g entsteht. (siehe Satz 69, 4.)
> Wenn h durch beschränkte Rekursion aus Funktionen niedrigerer Ordnung entsteht, dann ist $h \in \mathcal{E}^3$ und damit auch $\tau_H \in \mathcal{E}^3$.

Somit erhalten wir zunächst $\mathcal{E}^3 \subseteq \mathcal{E}^3_\tau$ und $\mathcal{E}^3 \subseteq \mathcal{E}^n_\tau$ für alle n, denn $\mathcal{E}^3 \subseteq \mathcal{E}^n$ impliziert direkt $\mathcal{E}^3_\tau \subseteq \mathcal{E}^n_\tau$ nach der Definition von $\mathcal{E}^n_\tau$.

Für die restlichen Klassen werfen wir nochmals einen Blick auf den Beweis von Satz 53. Dieser beschreibt eine primitiv-rekursive Definition von $f_{m+1}(x, y)$ aus $f_m(x, y)$. Die Schrankenfunktionen kann man dabei einfach weglassen und die berechnende Turingmaschine wird trotzdem zum selben Ergebnis kommen. Das ändert auch nicht den Fakt, dass alle Teilfunktionen aus der Konstruktion in $\mathcal{E}^n$ liegen. Folglich liegen alle diese

Teilfunktionen, einschließlich $f_{m+1}(x, y)$, auch in $\mathcal{E}^n_\tau$, denn für jede rekursive Teilfunktion $h(\vec{u}, y)$ gilt nach Satz 69:

$$\tau_H(\vec{u}, y) = 2 * \left(l(\vec{u}, y) + \max \left\{ \tau_F(\vec{u}),\ \max_{z \leq y} \{\tau_G(\vec{u}, z, h(\vec{u}, z))\} \right\} \right) \in \mathcal{E}^n$$

Die Definition von $f_{m+1}(x, y)$ benötigt abgesehen von $f_m(x, y)$ nur Funktionen aus $\mathcal{E}^3 \subseteq \mathcal{E}^n_\tau$. Nach Induktion enthält $\mathcal{E}^n_\tau$ also alle $f_m(x, y)$ für $m \leq n$.

Jetzt haben wir gezeigt, dass $\mathcal{E}^n_\tau$ alle Initialfunktionen von $\mathcal{E}^n$ enthält und können den Induktionsbeweis von oben wiederholen. Das Ergebnis davon ist $\mathcal{E}^n \subseteq \mathcal{E}^n_\tau$. □

Dieser Satz stellt also eine Verbindung zwischen den algebraisch definierten Klassen von Funktionen und den Turingmaschinen, die diese berechnen, her. Die Berechnung einer Funktion kann nicht für alle Eingaben schwieriger sein als die Berechnung einer anderen Funktion, die in der Grzegorczyk Hierarchie weiter oben liegt. Allerdings gilt dies nicht nur für Turingmaschinen, sondern prinzipiell für jedes Berechnungsmodell, welches sich innerhalb von $\mathcal{E}^2$ arithmetisieren lässt, solange man Laufzeit und Speicherbedarf auf passende Weise definiert. Zum Beispiel bleibt der Zusammenhang erhalten, wenn man Mehrband-Turingmaschinen erlaubt.

3.2 Der Grenzfall $\mathcal{E}^2$

Das Problem mit diesem Satz ist, dass er für $\mathcal{E}^2$ selbst nicht mehr funktioniert. $\mathcal{E}^2$ enthält bereits alle Polynome und einfachere Funktionen. Um den Unterschied zwischen Addition und Multiplikation zu finden, reicht Satz 71 nicht aus. Allerdings hat Ritchie noch einen weiteren Zusammenhang gefunden:

Satz 72 ([1, S. 27], vgl. [4, S. 149-153]). *Eine Funktion $f(x_1, \ldots, x_n)$ liegt in $\mathcal{E}^2$ genau dann, wenn f von einer linear-beschränkten Turingmaschine Z berechnet wird, also wenn es c_1 und c_2 gibt, sodass für alle $x_1, \ldots, x_n$ gilt:*

$$\tau_Z(x) \leq c_1 * l(x_1, \ldots, x_n) + c_2$$

Der Speicherbedarf wächst also linear bezüglich der Eingabelänge.

Beweis. Sei F' die Klasse der Funktionen, für die eine Turingmaschine Z existiert, sodass $\tau_Z(x) \leq c_1 * l(x_1, \ldots, x_n) + c_2$ gilt. Dann ist F' abgeschlossen unter Substitution und beschränkter Rekursion. Für alle g gilt $l(g(\vec{y})) \leq \tau_G(\vec{y})$, da mindestens die Ausgabe auf das Band geschrieben werden muss. Seien $f, g \in F'$ mit

$$\tau_F(\vec{x}) \leq c_F * l(\vec{x}) + c \quad \text{und} \quad \tau_G(\vec{y}) \leq c_G * l(\vec{y}) + c.$$

Daraus folgt nach Satz 69 etwas kurz gefasst:

$$\begin{aligned}
\tau_H(\vec{x}_1,\vec{x}_2,\vec{y}) &= l(\vec{x}_1,\vec{x}_2,\vec{y}) + \max\{\tau_F(\vec{x}_1,g(\vec{y}),\vec{x}_2),\, l(\vec{x}_1,\vec{x}_2)+\tau_G(\vec{y})\}\\
&\le l(\vec{x}_1,\vec{x}_2,\vec{y}) + \max\{c_F * l(\vec{x}_1,g(\vec{y}),\vec{x}_2)+c,\, l(\vec{x}_1,\vec{x}_2)+\tau_G(\vec{y})\}\\
&\le l(\vec{x}_1,\vec{x}_2,\vec{y}) + \max\{c_F * (l(\vec{x}_1,\vec{x}_2)+1+\tau_G(\vec{y}))+c,\, l(\vec{x}_1,\vec{x}_2)+\tau_G(\vec{y})\}\\
&\le l(\vec{x}_1,\vec{x}_2,\vec{y}) + c_F * (l(\vec{x}_1,\vec{x}_2)+1+\tau_G(\vec{y}))+c\\
&\le l(\vec{x}_1,\vec{x}_2,\vec{y}) + c_F * (l(\vec{x}_1,\vec{x}_2)+1+c_G * l(\vec{y})+c)+c\\
&\le (c_F+1)c_G * l(\vec{x}_1,\vec{x}_2,\vec{y}) + (c_F+1) * c
\end{aligned}$$

$$\begin{aligned}
\text{oder } \tau_H(\vec{x}_1,\vec{x}_2,\vec{x}_3,y) &= l(\vec{x}_1,\vec{x}_2,\vec{x}_3,y) + \tau_F(\vec{x}_1,y,\vec{x}_2,y,\vec{x}_3)\\
&\le l(\vec{x}_1,\vec{x}_2,\vec{x}_3,y) + c_F * l(\vec{x}_1,y,\vec{x}_2,y,\vec{x}_3)+c\\
&\le (2c_F+1) * l(\vec{x}_1,\vec{x}_2,\vec{x}_3,y)+c
\end{aligned}$$

$$\begin{aligned}
\text{oder } \tau_H(\vec{x}_1,\vec{x}_2) &= l(\vec{x}_1,\vec{x}_2) + \tau_F(\vec{x}_1,0,\vec{x}_2)\\
&\le l(\vec{x}_1,\vec{x}_2) + c_F * l(\vec{x}_1,0,\vec{x}_2)+c\\
&\le (c_F+1) * l(\vec{x}_1,\vec{x}_2)+c+3
\end{aligned}$$

Oder, wenn $h(\vec{u},y)$ durch $j(\vec{u},y)\in F'$ mit $\tau_J(\vec{u},y)\le c_J * l(\vec{u},y)+c$ beschränkt ist, gilt:

$$\begin{aligned}
\tau_H(\vec{u},y) &= 2 * \Big(l(\vec{u},y) + \max\Big\{\tau_F(\vec{u}),\, \max_{z\le y}\{\tau_G(\vec{u},z,h(\vec{u},z))\}\Big\}\Big)\\
&\le 2 * \Big(l(\vec{u},y) + \max\Big\{c_F * l(\vec{u})+c,\, \max_{z\le y}\{c_G * l(\vec{u},z,j(\vec{u},z))+c\}\Big\}\Big)\\
&\le 2 * \Big(l(\vec{u},y) + \max\Big\{c_F * l(\vec{u}),\, \max_{z\le y}\{c_G * (l(\vec{u},z)+l(j(\vec{u},z)))\}\Big\}+c\Big)\\
&\le 2 * \Big(l(\vec{u},y) + \max\Big\{c_F * l(\vec{u}),\, \max_{z\le y}\{c_G * (l(\vec{u},z)+c_J * l(\vec{u},z)+c)\}\Big\}+c\Big)\\
&\le 2 * \Big(l(\vec{u},y) + \max\Big\{c_F * l(\vec{u}),\, c_G(c_J+1) * l(\vec{u},z)+c_G * c\Big\}+c\Big)\\
&\le 2(c_F+c_G(c_J+2)) * l(\vec{u},y) + 2(c_G+1) * c
\end{aligned}$$

Die Projektionsfunktionen lassen sich in $l(x,y)+1$ Bandzellen berechnen und für $f_2(x,y)=(x+1)*(y+1)$ gibt eine Turingmaschine Z mit $\tau_Z(x,y)\le 2*l(x,y)+5$. Folglich ist

$$\mathcal{E}^2 = \Big[S(x), p_1^2, p_2^2, f_2(x,y)\,;\, Sub, lim.Rec\Big] \subseteq F'.$$

Da die Arithmetisierung einer Turingmaschine in $\mathcal{E}^2$ möglich ist, erhalten wir auch die Gegenrichtung. Sei also Z eine Turingmaschine mit s Zuständen, a Bandsymbolen und $\tau_Z(\vec{x})\le c_1 * l(\vec{x})+c_2$. Dann ist die Anzahl der möglichen Konfigurationen nicht größer

als

$$
\begin{aligned}
a^{\tau_Z(\vec{x})} * s * \tau_Z(\vec{x}) &\leq a^{2*\tau_Z(\vec{x})} * s \leq 2^{2*\tau_Z(\vec{x})*a} * s \leq 2^{(c_1*l(\vec{x})+c_2)*2a} * s \\
&= (2^{l(x_1,\dots,x_n)})^{2c_1 a} * 2^{2c_2 a} * s \leq \left(2^{n-1} * \prod_{i=1}^{n} 2^{l(x_i)}\right)^{2c_1 a} * 2^{2c_2 a} * s \\
&\leq \left(2^{n-1} * \prod_{i=1}^{n} (2x_i)\right)^{2c_1 a} * 2^{2c_2 a} * s = b_{n,z}(\vec{x}) \in \mathcal{E}^2
\end{aligned}
$$

Aus dieser Funktion $b_{n,z}(\vec{x})$ ergibt sich sowohl eine obere Schranke für die Laufzeit, als auch für die Gödelnummern der Konfigurationen. Es gilt also:

$$
\begin{gathered}
H_{n,z}(\vec{x}) \leq b_{n,z}(\vec{x}) \\
f(\vec{x}) = \text{Decode}\left(H_{n,z}\Big(\vec{x}, \mu(y \leq b_{n,z}(\vec{x}))\left[\text{Fin}\Big(H_{n,z}(\vec{x}, y)\Big)\right]\Big)\right) \in \mathcal{E}^2
\end{gathered}
$$

Damit liegt die Funktion, die Z berechnet, in $\mathcal{E}^2$ und wir haben $F' \subseteq \mathcal{E}^2$ gezeigt. Folglich gilt insgesamt $F' = \mathcal{E}^2$. □

Eine Folgerung aus diesem Satz ist, dass sich der Speicherbedarf in dieser Form nicht zur Klassifizierung der Funktionen in $\mathcal{E}^2$ eignet. Denn endlich viele Bandzellen lassen sich mit Hilfe eines größeren Alphabets immer in einem Bandsymbol zusammenfassen. Da der Speicher hier mit einem konstanten Faktor beschränkt ist, kann für jede Funktion in $\mathcal{E}^2$ eine Turingmaschine konstruiert werden, die die Berechnung allein im Speicher der Eingabe durchführt. Nur die Ausgabe benötigt eventuell mehr Platz.

3.3 Polynomialzeit

Da Cobham den Speicherbedarf als Metrik für einfache Funktionen erstmal ausgeschlossen hatte, wandte er sich der Laufzeit zu. Die Anzahl der benötigten Schritte zur Addition zweier Zahlen wächst linear mit der Länge der beiden Zahlen. Um zwei Zahlen zu multiplizieren, benötigt man für jede Ziffer der ersten Zahl in b-adischer Darstellung höchstens b Additionen. Die Summanden werden dabei durch einen Linksshift erzeugt, also indem an die Zahl eine 0 angehängt wird. Hier wächst also die Anzahl der Additionen linear mit der Länge der Zahlen und damit ist die Gesamtlaufzeit quadratisch.
Sowohl bei Addition als auch bei Multiplikation ist Laufzeit also durch ein Polynom beschränkt. Diese Eigenschaft haben viele relativ einfache Funktionen, wie zum Beispiel auch die Division und die Quadratwurzel. Daraus entstand die Funktionsklasse $\mathcal{L}$. Dies war eine der ersten Erwähnungen von Polynomialzeit in der Komplexitätstheorie.

Definition 73 ([1, S. 28]). $\mathcal{L}$ ist die Klasse aller Funktionen, die von einer Turingmaschine Z berechnet werden, sodass σ_Z von einem Polynom P in der Länge der Argumente beschränkt ist, also:

$$\sigma_Z(\vec{x}) \leq P(l(\vec{x})) \text{ für ein } P(X) \in \mathbb{Z}[X]$$

Da alle Polynome in $\mathcal{E}^2$ liegen, könnte man vermuten, dass $\mathcal{L}$ und $\mathcal{E}^2$ ähnliche Klassen beschreiben. Aus Satz 71 wissen wir aber nur: $\mathcal{L} \subseteq \mathcal{E}^3$. $\mathcal{L}$ hat zwar einige interessante Abschlusseigenschaften, aber diese passen nicht ganz zur Grzegorczyk Hierarchie.

Definition 74 ([vgl. 1, S. 28]). Eine Funktionsklasse heißt abgeschlossen unter der **Operation der beschränkten Rekursion auf b-adischer Notation**, wenn sie für alle Funktionen $f(\vec{u})$, $g_0(\vec{u}, x, y), \ldots, g_{b-1}(\vec{u}, x, y)$ und $h(\vec{u}, x)$ auch die Funktion $F(\vec{u}, x)$ enthält, sodass für alle $\vec{u}$, x und $i < b$ gilt:

$$\begin{aligned} F(\vec{u}, 0) &= f(\vec{u}) \\ F(\vec{u}, s_i^b(x)) &= g_i(\vec{u}, x, F(\vec{u}, x)) \\ F(\vec{u}, x) &\leq h(\vec{u}, x) \end{aligned}$$

s_i^b beschreibt hier die Konkatenation mit der Ziffer i in b-adischer Notation, also gilt:

$$s_i^b(x) = b * x + i$$

Die Rekursion durchläuft also einen Schritt für jede Ziffer von x. Unten wird diese Operation bezeichnet mit $lim.Rec.Not(b)$.

Satz 75 ([vgl. 1, S. 28]). *Die Klasse $\mathcal{L}$ ist abgeschlossen unter Substitution und beschränkter Rekursion auf b-adischer Notation für alle b.*

Beweis. Der Beweis folgt dem Prinzip von Satz 69 mit Laufzeit anstelle von Speicher. Die Substitution lässt sich genau so in Polynomialzeit umsetzten. Rekursion funktioniert aber nur in Polynomialzeit, wenn die Rekursionstiefe polynomiell durch die Eingabelänge beschränkt ist. □

Da das Verhältnis zwischen Länge und Wert einer Zahl exponentiell ist, lässt sich primitive Rekursion im allgemeinen nicht in Polynomialzeit umsetzen.
Cobham hat sogar eine vollständige Charakterisierung der Klasse $\mathcal{L}$ angegeben:

Satz 76 ([vgl. 1, S. 28]). *Für alle b gilt:*

$$\mathcal{L} = \left[s_0^b(x), \ldots, s_{b-1}^b(x), x^{l(y)} \,;\, Sub, lim.Rec.Not(b) \right]$$

Beweis. Die s_i^b lassen sich bei b-adischer Darstellung in linearer Zeit berechnen. Da Multiplikation in Polynomialzeit möglich ist, liegt auch die $l(y)$-fache Multiplikation von x mit sich selbst, also $x^{l(y)}$, in $\mathcal{L}$. Mit Satz 75 folgt dann:

$$\left[s_0^b(x), \ldots, s_{b-1}^b(x), x^{l(y)}\,;\, Sub, lim.Rec.Not(b)\right] \subseteq \mathcal{L}.$$

Die andere Seite des Beweises wird hier nicht ausgeführt. Dazu müsste man zeigen, dass Turingmaschinen auch in $\left[s_0^b(x), \ldots, s_{b-1}^b(x), x^{l(y)}\,;\, Sub, lim.Rec.Not(b)\right]$ arithmetisiert werden können, was den Rahmen dieser Arbeit überschreiten würde. □

Interessant an der Charakterisierung von $\mathcal{L}$ ist, dass die Funktion $x^{l(y)}$ nicht in $\mathcal{E}^2$ liegt, denn $l(x^{l(y)}) \geq (l(x)-1) * l(y)$ ist nicht linear beschränkt durch $l(x,y)$ und kann somit nach Satz 72 nicht in $\mathcal{E}^2$ liegen. Cobhams Vermutung ist, dass sich die beiden Klassen $\mathcal{L}$ und $\mathcal{E}^2$ nicht vergleichen lassen. In diesem Fall wäre also die Charakterisierung relativ einfacher Funktionen anhand von Laufzeit und Speicher eine gänzlich verschiedene. Als Kriterium dafür verwies Cobham auf die Funktion prim(k), die die k-te Primzahl berechnet und von der er vermutete, dass sie nicht in $\mathcal{L}$ liegt. Diese Frage ist bis heute offen geblieben. Anhand von Definition 26 wird aber deutlich, dass prim(k) in $\mathcal{E}^2$ liegt. Um herauszufinden, ob Addition tatsächlich einfacher ist als Multiplikation, müsste man $\mathcal{L}$ in weitere Unterklassen teilen. Die offensichtliche Aufteilung wäre, mit $\mathcal{L}^n$ die Funktionen zu bezeichnen, deren Laufzeit durch ein Polynom von Grad n beschränkt wird. Dabei stößt man allerdings auf ein Problem. Die Laufzeiten innerhalb von $\mathcal{L}$ sind stark abhängig von dem zugrundeliegenden Maschinenmodell. Auf einer Turingmaschine mit nur einem Band ist Addition nicht in linearer Zeit möglich, da der Kopf für jede Ziffer zwischen den beiden Summanden wechseln muss, was bereits lineare Zeit benötigt. Demnach läge Addition hier nicht in $\mathcal{L}^1$. Stehen jedoch zwei Bänder zur Verfügung, kann man die erste Zahl kopieren, sodass in der weiteren Rechnung beide Argumente gleichzeitig gelesen werden können. Die Addition liegt also doch in $\mathcal{L}^1$. Der gleiche Zusammenhang besteht zwischen der Multiplikation und $\mathcal{L}^2$. Wenn man also eine algebraische Charakterisierung von Klassen $\mathcal{L}^n$ vornehmen will, braucht man ein geeignetes Maschinenmodell, welches sich in $\mathcal{L}^1$ arithmetisieren lässt. Und selbst, wenn man eine solche Maschine konstruiert hätte, bliebe immer noch die Frage, ob die resultierende Einteilung von $\mathcal{L}$ auch über echten Maschinen als sinnvoll erscheint.
Eine weitere Möglichkeit der Charakterisierung wäre, ein anderes Maß an Stelle von Laufzeit und Speicher zu wählen. Der Begriff der Effizienz hängt häufig von einem konkreten Anwendungsfall ab und wenn man Funktionen anhand von verfügbaren Algorithmen vergleicht, kann man trotzdem entscheiden, welcher von ihnen gerade am besten geeignet ist. Aber solange keine zufriedenstellende Formalisierung einer $\mathcal{L}^n$ Hierarchie gefunden wurde, wird man wohl nicht beweisen können, dass die Addition im allgemeinen Sinne einfacher ist als die Multiplikation.

Zusammenfassung

Mit den algebraischen Hüllenoperatoren haben wir ein Konzept beschrieben, das unendlichen Mengen von Funktionen eine einfache Struktur verleiht. Mit vollständiger Induktion auf der Ordnung der Elemente lassen sich dadurch Aussagen über die ganze Menge beweisen. Wenn man zwei Mengen mit ähnlichen Charakterisierungen betrachtet, lässt sich daraus leicht eine Teilmengenbeziehung ableiten. Grzegorczyk hat davon Gebrauch gemacht, indem er so alle primitiv-rekursiven Funktionen klassifiziert hat. Die Grzegorczyk Hierarchie besteht aus einer Folge von Funktionsklassen $\mathcal{E}^n$, in der jede Klasse Funktionen mit schnellerem Wachstum enthält als ihr Vorgänger. In $\mathcal{E}^0$ sind Funktionen höchstens um eine Konstante größer als ihre Argumente. $\mathcal{E}^1$ führt die Addition ein. $\mathcal{E}^2$ enthält die Multiplikation und $\mathcal{E}^3$ enthält x^y. In den weiteren Klassen wird diese Konstruktion ins unendliche fortgesetzt.
Ritchie hat erkannt, dass sich aus dem beschränkten Wachstum auch Komplexitätsklassen auf den Turingmaschinen ergeben, die die Funktionen berechnen. Er hat dabei Turingmaschinen innerhalb von $\mathcal{E}^2$ arithmetisiert. Für $n > 2$ sind alle Klassen $\mathcal{E}^n$ äquivalent zu den Klassen von Funktionen, die von Turingmaschinen berechnet werden, deren Laufzeit oder Speicherbedarf in $\mathcal{E}^n$ liegt. Formalisiert wurde dieser Zusammenhang von Cobham. Für die Klasse $\mathcal{E}^2$ selbst konnte er allerdings nur feststellen, dass sie eine Funktion genau dann enthält, wenn diese von einer linear beschränkten Turingmaschine berechnet wird. Eine ähnliche Klasse bezüglich der Laufzeit definierte Cobham durch die Klasse der Polynomialzeitfunktionen. Ihre algebraische Charakterisierung ist jedoch eine andere als bei $\mathcal{E}^2$ und so kam er zu der Vermutung, dass sich diese beiden Klassen nicht vergleichen lassen.
Die Klasse der Polynomialzeit ist seitdem ein zentrales Objekt in den Untersuchungen der Komplexitätstheorie geworden und auch das Konzept der algebraischen Charakterisierung von Komplexitätsklassen findet dabei heutzutage immer noch Anwendungen.

Quellenverzeichnis

[1] Alan Cobham. „The Intrinsic Computational Difficulty of Functions“. eng. In: *Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress (Studies in Logic and the Foundations of Mathematics)*. Hrsg. von Yehoshua Bar-Hillel. North-Holland Publishing, 1965, S. 24–30.

[2] Andrzej Grzegorczyk. *Some classes of recursive functions*. eng. Warszawa: Instytut Matematyczny Polskiej Akademi Nauk, 1953.

[3] Rózsa Péter. „Über den Zusammenhang der verschiedenen Begriffe der rekursiven Funktion“. In: *Mathematische Annalen* 110.1 (Dez. 1935), S. 612–632. ISSN: 1432-1807. DOI: 10.1007/BF01448046.

[4] Robert W. Ritchie. „Classes of predictably computable Functions“. In: *Transactions of the American Mathematical Society* 106 (1963), S. 139–173.

[5] Raphael M. Robinson. „Primitive Recursive Functions“. eng. In: *Journal of Symbolic Logic* 13.2 (1948), S. 113–114. DOI: 10.2307/2267333.

[6] Klaus W. Wagner. „Mathematische Grundlagen“. In: *Theoretische Informatik: Eine kompakte Einführung*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, S. 5–20. ISBN: 978-3-642-55452-0. DOI: 10.1007/978-3-642-55452-0_2.