Blum-Schub-Smale Rechenmodell: Motivation, Definition, Beispielalgorithmen

Seminar Reelle Komplexität: Dr. habil. Ulrike Brandt, Prof. Dr. Ziegler
Name: Yevgen Chebotar
Sommersemester 2011

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1 Motivation

In dem Buch *Complexity and Real Computation*[1] beschreiben die Autoren ein neues Rechenmodell, das für die Modellierung von vielen wissenschaftlichen und insbesondere matematischen Algorithmen ein geeignetes Framework darstellt.

Die Autoren weisen darauf hin, dass das Turing-Modell aus der klassischen Berechnungstheorie zu sehr von den Zahlen aus der Menge $\{0, 1\}$ abhängt, wobei der größte Teil wissenschaftlicher Algorithmen mit reellen Zahlen arbeitet, z.B. Newton-, Euler- und Gauss- Algorithmen. Somit ist das Turing-Modell unzulänglich für die reelle Berechnungen. John von Neumann bestätigt dies und erklärt, dass das 1/0 bzw. *Alles-oder-nichts* Konzept sehr wenig mit der kontinuierlichen Natur der reellen und komplexen Zahlen in Berührung kommt. Deshalb bleibt die klassische Turing-Theorie von der mathematischen Analysis abgeschnitten, wobei die Analysis ein der erfolgreichsten und weitentwickelsten Teilen von Mathematik ist. Dagegen wird in der klassischen Theorie sehr oft Kombinatorik verwendet, die zu einem der schwierigsten mathematischen Gebieten gehört.

Mit der Einführung von Blum-Schub-Smale (BSS) Rechenmodell wollen die Autoren eine Theorie von reellen Berechnungen entwickeln, die weniger auf der Logik und mehr auf den mathematischen Verfahren basiert. Dennoch behält neue Theorie die Turing-Theorie als ein Spezialfall und somit enthält eine Verbindung zu der klassichen Berechnungstheorie.

Obwohl das BSS-Modell in der Realität nicht implementiert werden kann, kann es jedoch hilfreich sein, um untere Komplexitätsgrenzen von mathematischen Algorithmen zu bestimmen. Wenn ein Problem in diesem Modell nicht effizient lösbar ist, dann wäre es noch schwieriger dieses Problem in realen Systemen zu lösen.

2 Beispielprobleme

Im Folgenden wird eine Reihe von mathematischen Problemen dargestellt, die zur Betrachtung und Definition von dem BSS-Modell hilfreich sein können.

2.1 Mandelbrot-Menge

Die Mandelbrot-Menge wurde nach dem Mathematiker *Benoît Mandelbrot* gennant. Um diese Menge zu bestimmen, definieren wir zuerst eine Funktion über komplexe Zahlen, wobei $c \in \mathbb{C}$ konstant ist:

$$p_c : \mathbb{C} \to \mathbb{C}, z \mapsto z^2 + c, c \in \mathbb{C}$$

Durch eine mehrfache Anwendung dieser Funktion können wir eine Folge erstellen, wobei n-tes Glied dieser Folge durch n-fache Anwendung der Funktion entsteht: $p_c^n(z) = p_c(...p_c(p_c(p_c(z))))$. Betrachten wir die Werte der Folge für $z = 0$:

$$p_c(0) = c, p_c^2(0) = c^2 + c, p_c^3(0) = (c^2 + c)^2 + c, ...$$

Die Mandelbrot-Menge $\mathbf{M}$ enthält alle komplexen Zahlen c, für welche die oben beschriebene Folge für $z = 0$ immer beschränkt bleibt, d.h.:

$$\mathbf{M} = \{c \in \mathbb{C} \mid \exists s \in \mathbb{R}\ \forall n \in \mathbb{N}\ |p_c^n(0)| \leq s\}$$

Der Komplement $\mathbf{M}'$ von der Mandelbrot-Menge enthält alle komplexe Zahlen, für welche gilt:

$$\mathbf{M}' = \{c \in \mathbb{C} \mid p_c^n(0) \to \infty \text{ für } n \to \infty\}$$

[1] Blum, Lenore/Cucker, Felipe/Schub, Michael/Smale, Steve: Complexity and Real Copmutation, New York: Springer 1997

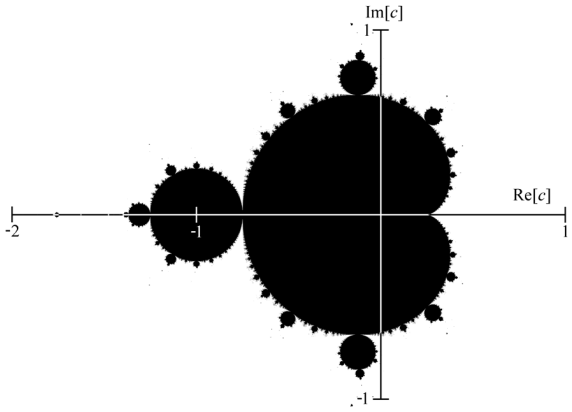


Abbildung 2.1: Die Zahlen innerhalb des schwarzen Bereiches liegen in der Mandelbrot-Menge.

Abbildung 2.1 zeigt die Mandelbrot-Menge in der komplexen Zahlenebene[1], wobei die Zahlen aus der Menge in schwarzer Farbe eingezeichet sind. Wie wir sehen, handelt es sich um einer sehr komplexen Struktur. Roger Penrose stellt in seinem Buch *The Emperor's New Mind*[2] die Frage ob diese Menge entscheidbar ist. Die Menge ist entscheidbar wenn es für jede Zahl möglich ist zu bestimmen, ob diese Zahl zu der Menge gehört oder nicht.

Penrose bemerkt, dass für das Lösen von dieser Frage mit Hilfe von der klassischen Theorie die Menge abzählbar sein muss. Mandelbrot-Menge ist aber überabzählbar, da sie eine Untermenge von komplexen Zahlen ist. Mit Hilfe von der rekursiven Analyse ist es jedoch möglich nicht abzählbare reelle Zahlen (und somit auch komplexe Zahlen) in der Turing-Maschine als Limit von der sequentiellen Ausgabe darzustellen. Die Maschine wird die Zahl Bit für Bit ausgeben und somit die Genauigkeit dieser Zahl immer vergrößern. Bei unendlich vielen Bits entspricht die Ausgabe dem gesuchten reellen Zahl. Es ist jedoch unmöglich mit dieser Methode die Gleichheit zweier reellen Zahlen nachzuweisen, da jedes weiteres ausgegebenes Bit von zwei Zahlen unterschiedlich sein kann und die Gleichheitsaussage nie genau beantwortet wird. Die Prüfung von der Gleichheit und insbesondere von der Ungleichheit von reellen Zahlen ist aber unabdingbar für die Frage der Entscheidbarkeit der Mandelbrot-Menge. Das BSS-Rechenmodell stellt ein theoretisches Modell einer algebraischen Maschine dar. Diese Maschine ist in der Lage direkt mit reellen und komplexen Zahlen zu operieren und kann dadurch solche Gleichheitsfragen behandeln.

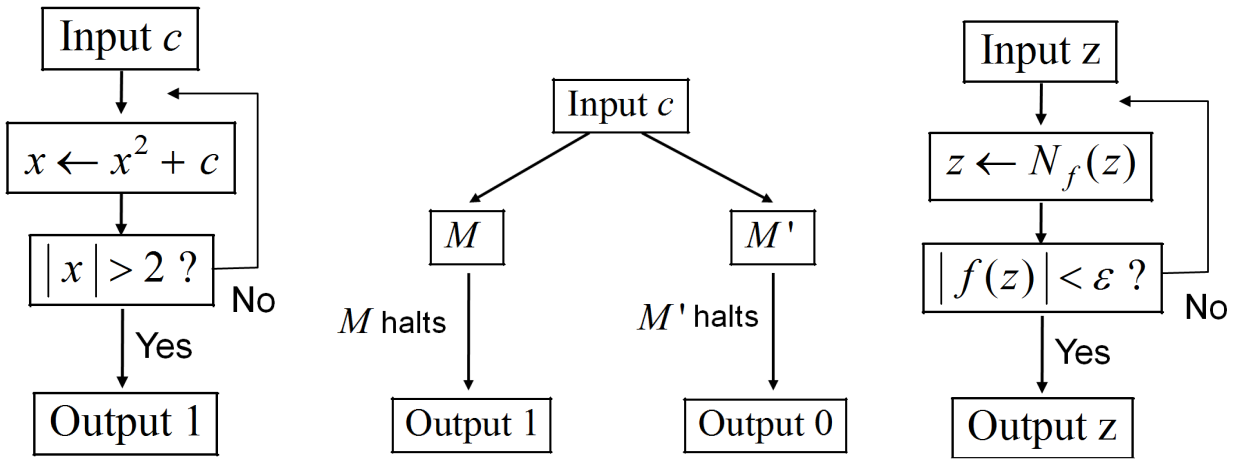
Es ist bekannt dass wenn der Betrag eines Elements der Folge $c, c^2+c, p(c^2+c)^2+c, \ldots$ jemals größer 2 wird, dann wird die Folge gegen unendlich divergieren. Somit würde die Zahl c zu der Folge $\mathbf{M}'$ gehören. Die Abbildung 2.2a zeigt eine Maschine die in der Lage wäre diese Bedingung so lange zu überprüfen bis sie erfüllt ist. Die Maschine hat einen Verzweigungsknoten. Sie gibt **1** zurück falls $c \in \mathbf{M}'$. Wenn es nicht der Fall ist, dann hält die Maschine nicht. Somit ist die Menge $\mathbf{M}'$ semi-entscheidbar. Wenn die Mandelbrot-Menge $\mathbf{M}$ auch semi-entscheidbar wäre, dann könnten wir eine Maschine konstruieren, die in der Abbildung 2.2b abgebildet ist. Diese Maschine verbindet zwei Maschinen miteinander: die Maschine M', die wenn $c \in \mathbf{M}'$ hält; die Maschine M, die wenn $c \in \mathbf{M}$ hält. Laut der Definition, wäre dann die Mandelbrot-Menge $\mathbf{M}$ entscheidbar. Ein der Ziele des BSS-Modells ist es die Existenzfrage einer solcher Maschine beantworten zu können.

2.2 Newton-Verfahren

Das Newton-Verfahren ist ein Nullstellen-Suchalgorithmus. Wenn wir das Vefahren für die komplexe Zahlen betrachten, dann sucht das Newton-Algorithmus für eine gegebene Funktion f iterativ eine Zahl $z \in \mathbb{C}$, so dass $f(z) = 0$. Der Vorschrift für die Newton-Iteration ist ein Endomorphismus $N_f : \mathbb{C} \to \mathbb{C}$:

$$N_f(z) = z - \frac{f(z)}{f'(z)},\ f'(z) \neq 0$$

[1] Bildquelle: http://commons.wikimedia.org/wiki/File:Mandelset_hires.png
[2] Penrose, Roger: The Emperor's New Mind, Oxford: Oxford University Press 1989



(a) Algorithmus für $c \in \mathbf{M}'$ (b) Entscheidungsmaschine für die Mandelbrot-Menge (c) Newton-Maschine

Abbildung 2.2: Entscheidungsmaschinen für die Mandelbrot-Menge und für das Newton-Verfahren

Wenn wir einen Startpunkt z_0 auswählen, dann wird durch das mehrfache Anwenden von N_f eine Folge generiert, wobei der nächste Folgenglied wie folgt berechnet werden kann:

$$z_{k+1} = N_f(z_k) = N_f^{k+1}(z_0).$$

Bei einem "gut" ausgewählten Startpunkt konvergiert die Folge gegen eine Nullstelle von f. Es ist möglich, dass die Folge die Nullstelle immer weiter annähert aber nie genau erreicht. Deswegen ist es nötig einen Genauigkeitswert ε zu definieren, so dass wenn für irgendeine Iteration $|f(z_k)| < \varepsilon$ wird, dann wird die Nullstellen-Suche abgebrochen und z_k zurückgegeben. Abbildung 2.2c zeigt wie eine Maschine für das Newton-Algorithmus aussehen könnte.

Um die Konvergenzgeschwindigkeit und somit auch die Komplexität des Newton-Verfahrens zu analysieren, können wir zwei seine Eigenschaften betrachten:

1. $f(\zeta) = 0$ gdw. $N_f(\zeta) = \zeta$. Beim Einsetzen von $f(\zeta)$ in das Newton-Endormophismus müssen wir beachten, dass $f'(z) \neq 0$. Es ist jedoch möglich Polynome g und h zu finden, so dass $f/f' = g/h$ und für $f(\zeta) = 0$ gilt, dass $g(\zeta) = 0$ und $h(\zeta) \neq 0$. Somit gilt:

$$N_f(\zeta) = \zeta - \frac{f(\zeta)}{f'(\zeta)} = \zeta - \frac{g(\zeta)}{h(\zeta)} = \zeta - 0 = \zeta$$

Die Nullstellen von f sind also die Fixpunkte von N_f

2. Wenn $N_f(\zeta) = \zeta$ dann $|N_f'(\zeta)| < 1$. Bestimmen wir zuerst $N_f'(z)$:

$$N_f'(z) = 1 - \frac{f'(z) \cdot f'(z) - f(z) \cdot f''(z)}{f'(z)^2} = \frac{f'(z)^2}{f'(z)^2} - \frac{f'(z)^2 - f(z) \cdot f''(z)}{f'(z)^2} = \frac{f(z) \cdot f''(z)}{f'(z)^2}$$

Betrachten wir die Taylorentwicklung von f um die Stelle ζ für $a_m \neq 0$:

$$f(z) = a_m(z - \zeta)^m + \text{Terme von höherer Ordnung}$$

Setzen wir diese in N_f' (die höheren Terme können ausgelassen werden). Berechnen wir zuerst f' und f'':

$$f'(z) = a_m m(z - \zeta)^{m-1}, \quad f''(z) = a_m m(m-1)(z - \zeta)^{m-2}$$

Einsetzen in N_f' ergibt:

$$|N_f'(z)| = \frac{(a_m(z-\zeta)^m)(a_m m(m-1)(z-\zeta)^{m-2})}{(a_m m(m-1)(z-\zeta)^{m-1})^2} = \frac{a_m^2 m(m-1)(z-\zeta)^{2m-2}}{a_m^2 m^2 (z-\zeta)^{2m-2}} = \frac{m-1}{m}$$

Da $m > 0$ gilt $|N_f'(\zeta)| < 1$. Die Nullstellen von f sind also anziehende Fixpunkte von N_f. Wenn zusätzlich ζ eine einfache Nullstelle von f ist, dann $m = 1$ und $|N_f'(\zeta)| = \frac{1-1}{1} = 0 \rightarrow N_f'(\zeta) = 0$. Der Fixpunkt ist in diesem Fall super-anziehend und wir beobachten eine quadratische Konvergenz.

Zusätzlich zu der Komplexität des Newton-Verfahrens, können wir in diesem Zusammenhang ein Entscheidungsproblem definieren. Es ist möglich, dass Newton-Iteration nicht für alle Startpunkte z_0 zu einer Nullstelle konvergiert. Insbesondere kann alternierendes oder periodisches Verhalten auftreten, wo die Iterationswerte sich ständig wiederholen (z.B. 1,0,1,0 ...). Somit entsteht die Frage, ob wir immer entscheiden können, dass ein Punkt ein guter Startpunkt ist, d.h. zur Konvergenz des Verfahrens führt. Die Bestimmung der Entscheidbarkeit von der Menge der guten Startpunkte könnte eine der Aufgabe des BSS-Rechenmodells sein.

2.3 Rucksackproblem

Intuitiv kann das Rucksackproblem mit einer folgenden Situation erklärt werden:

- Wir haben einen Rucksack, dass ein bestimmtes Volumen hat, z.B. 1 Liter. Weiterhin haben wir eine Menge von Gegenständen, die wir in den Rucksack legen können. Jeder Gegenstand hat ein eigenes Volumen. Wir suchen nach einer Untermenge dieser Gegenständen, so dass wenn wir sie in den Rucksack legen er genau voll gefüllt wird.

Für eine formale Definition nehmen wir an, dass die Elemente, die in den Rucksack gelegt werden, zu einem kommutativen Ring R mit Einheit gehören (z.B. $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$). Die Menge der Gegenständen, die eine Untermenge hat, die den Rucksack voll ausfühlt, kann mit Annahme, dass das Kapazität des Rucksacks 1 ist, folgendermassen definiert werden:

$$K_n = \{x \in R^n \mid \exists b \in \{0,1\}^n \text{ so dass } \sum_{i=0}^{n} b_i x_i = 1\}$$

Dabei ist $b \in \{0,1\}^n$ eine gültige Auswahl oder Belegung von Gegenständen. Wenn n-tes Element von b **1** ist, dann wird der n-te Gegenstand in den Rucksack gelegt, bei 0 wird er ausgelassen.

Es ist bekannt, dass das Rucksackproblem entscheidbar ist. Wir können einfach über alle Belegungen $b \in \{0,1\}^n$ iterieren. Falls eine gültige Belegung gefunden wird, dann geben wir 1 zurück. Andernfalls wird 0 zurückgegeben. Im Worst-Case müssen wir 2^n Belegungen überprüfen. Wenn wir aber eine Belegung haben, dann ist es leicht ihre Gültigkeit zu überprüfen. In der klassischen Theorie ist das Rucksackproblem über $\mathbb{Z}_2$ NP-vollständig. Es entsteht die Frage ob es allgemein für jeden Ring R gilt.

Das Rucksackproblem kann in ein algebraisches Problem umgeformt werden. Die Frage ist ob die Multiplikation, dass in dem neuen Problem verwendet wird, zu einem beschleunigten Algorithmus führen kann. Betrachten wir ein Polynom $k_n(x)$:

$$k_n(x) = \prod_{b \in \{0,1\}^n} (\sum_{i=0}^{n} b_i x_i - 1)$$

Wenn b eine gültige Belegung ist, dann $\sum_{i=0}^{n} b_i x_i = 1$. Somit ist $\sum_{i=0}^{n} b_i x_i - 1 = 0$ und $k_n(x) = 0$. Die Menge K_n ist dann equivalent zu:

$$V_{k_n} = \{x \in R^n \mid k_n(x) = 0\}$$

Das Rucksackproblem kann also durch eine Nullstellensuche des Polynoms $k_n(x)$ gelöst werden.

2.4 Hilbert-Nullstellensatz

Bei dem Hilbert Nullstellensatz fragen wir nach der Existenz einer gemeinsamen Nullstelle von mehreren Polynomen. Betrachten wir das Problem über die komplexen Zahlen (HN/$\mathbb{C}$). Gegeben sind die Polynome $f = \{f_1, ..., f_k\}$ mit n Variablen über $\mathbb{C}$. Wir suchen einen Vektor $\zeta \in \mathbb{C}^n$, der die Werte von allen Variablen enthält, so dass alle Polynome gleich Null werden, d.h. $f_i(\zeta) = 0$ für alle i.

Wir suchen ein algebraisches Algorithmus, das dieses Problem behandlen könnte. Um eine Komplexitätsbetrachtung eines solchen Algorithmus durchführen zu können, brauchen wir die Eingabegröße und den Komplexitätsmaß.

Die Eingabe des Algorithmus sind die Koeffizienten von allen Polynomen. Um die Anzahl N dieser Koeffizienten zu berechnen, können wir den Binomialkoeffizient verwenden, der alle Permutationen der möglichen Polynomglieder berücksichtigt. Die Anzahl der Koeffizienten eines Polynoms vom Grad d mit n Variablen:

$$\binom{n+d}{n}, \text{ z.B. für ein Polynom vom Grad 2 mit 2 Variablen: } \binom{2+2}{2} = 6$$

Das Polynom hat folgende Form:

$$a_1 + a_2 z_1 + a_3 z_2 + a_4 z_1^2 + a_5 z_2^2 + a_6 z_1 z_2$$

Für k Polynomen ist die gesamte Anzahl der Koeffizienten und somit die Größe der Eingabe $S(f)$:

$$N = \sum_{i=1}^{k} \binom{n+d}{n}$$

Als Komplexitätsmaß für das algebraische Algorithmus nehmen wir die Anzahl der arithmetischen Operationen $\mathcal{A}(f)$, die für die Ausführung des Algorithmus nötig sind. Die Maschine für dieses Algorithmus kann aufgrund einer von Hilbert bewiesener Eigenschaft der Polynome konstruiert werden. Es gibt keine gemeinsame Nullstelle falls es Polynome $g_1, ..., g_k$ mit n Variablen existieren, so dass:

$$\sum_{i=1}^{k} g_i f_i = 1$$

Die resultierende algebraische Maschine würde im Unterschied zu den Maschinen für die Mandelbrot-Menge und für das Newton-Verfahren nicht einen Verzweigungsknoten mit Ungleichheit-, sondern mit einer Gleichheitsabfrage haben, da die Menge der komplexen Zahlen nicht geordnet ist.

Die Koeffizienten von g_i kann man mit Hilfe von dem Gauß-Eliminationsverfahren bestimmen. Die Anzahl der dafür benötigten Schritte wächst exponentiell mit der Eingabegröße $S(f)$. Somit entsteht eine Vermutung dass es kein Algorithmus gibt, der HN/$\mathbb{C}$ in polynomieller Anazhl der Arithmetischen Operationen $A(f) \leq S(f)^c$ löst. Bei einem gegebenen Vektor $\zeta \in \mathbb{C}^n$ ist es jedoch für alle Polynome möglich zu überprüfen ob $f_i(\zeta) = 0$, und zwar in polynomieller Anzahl der arithmetischen Operationen. Folglich ist es sinnvoll zu vermuten, dass HN/$\mathbb{C}$-Problem NP-vollständig in $\mathbb{C}$ ist.

Das HN-Problem kann auch über andere Ringe betractet werden. Aus der klassischen Theorie ist es bekannt, dass HN/$\mathbb{Z}_2$ NP-vollständig ist. Die Frage ob P≠NP kann aus der klassischen Theorie auch auf die Mengen $\mathbb{C}$ und $\mathbb{R}$ übertragen werden. Der algebraische Ansatz, der für diese Mengen verwendet wird, kann durch die Verwendung verschiedener mathematischer Werkzeuge zu neuen Einblicken in das klassische Problem führen.

Das Problem der Suche nach gemeinsamen Nullstellen von mehreren Polynomen kann für Polynome über $\mathbb{R}$ in ein Problem überführt werden, in der die Nullstelle nur für ein Polynom gesucht wird. Betrachten wir die Polynome $f_1, ..., f_2$ über $\mathbb{R}$. Dann können wir ein Polynom g definieren, so dass:

$$g = \sum_{i=1}^{k} f_i^2$$

Dann gilt: $g(\xi) = 0$ gdw. $f_i(\xi) = 0$ für alle i. Da f_i zum Quadrat genommen wird, sind die Fälle, bei denen die Summe der Polynome durch verschiedene Vorzeichen zur Null wird, ausgeschlossen. Z.B. betrachten wir das Problem mit einer Variable und zwei Polynomen $f_1 = x$ und $f_2 = x - 2$. Für $x = 1$ gilt: $f_1(1) = 1$ und $f_2(1) = -1$, somit ist $f_1(1) + f_2(1) = 0$, obwohl $x = 1$ keine gemeinsame Nullstelle ist. Durch das Quadrieren erhalten wir: $f_1(1)^2 + f_2(1)^2 = 2$, also keine Nullstelle von g. Die Nullstellensuche für ein Polynom kann mit einem Algorithmus gelöst werden, das auf Ungleichheitsvergleichen verzweigt (vgl. Newton-Verfahren). Das Problem für Polynomen mit vier Variablen kann als 4-FEAS (engl. *Feasibility*) bezeichnet werden. Mit Hilfe von BSS-Rechenmodell kann es bewiesen werden, dass 4-FEAS NP-vollständig in $\mathbb{R}$ ist.

2.5 Lineare Programmierung

Bei der linearen Programmierung handelt es sich um die Suche nach Variablenbelegungen, die mehrere Ungleichungen gleichzeitig erfüllen. Betrachten wir eine Matrix $A \in \mathbb{R}^{m,n}$, Vektoren $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ und folgende Ungleichungen:

$$\begin{array}{cccc} a_{11}x_1 & +\ldots & +a_{1n}x_n & \geq b_1 \\ a_{21}x_1 & +\ldots & +a_{2n}x_n & \geq b_2 \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1}x_1 & +\ldots & +a_{mn}x_n & \geq b_m \end{array}$$

$$A = \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \ldots & a_{mn} \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

In Kurzschreibweise: $Ax \geq b$. Wir müssen entscheiden ob es solche x gibt, so dass alle Ungleichungen erfüllt sind. Alternativ kann auch lineare Programmierung mit Ungleichungen von der Form $Ax \leq b$ definiert werden. Das Problem über $\mathbb{R}$ bezeichnen wir als LPF/$\mathbb{R}$ (engl. *Linear Programming Feasibility*). Zusätzlich kann es auf ein Optimisierungsproblem erweitert werden. Dazu betrachten wir $c \in \mathbb{R}^n$ und versuchen ein x zu finden, so dass $c \cdot x$ minimal wird, unter der Bedingung, dass die Ungleichungen $Ax \geq b$ immer noch erfüllt bleiben. Alternativ können wir eine Maximisierung durchführen, in dem wir ein x suchen, so dass $c \cdot x$ maximal wird. Dieses Problem können wir als LPO/$\mathbb{R}$ (engl. *Linear Programming Optimization*) bezeichnen.

Durch Ersetzen von reellen Zahlen mit ganzen oder rationalen Zahlen definieren wir die entsprechenden Probleme IPF (engl. *Integer Programming Feasibility*) und IPO (engl. *Integer Programming Optimization*). Im Unterschied zu $\mathbb{R}$ ist es für die ganzen Zahlen möglich, dass die Ungleichungen nur reelle aber keine ganzzahlige Lösungen haben.

Betrachten wir die Komplexität der Programmierung- und Optimisierungsproblemen. Die Eingabe von LPF besteht aus der Matrix A und dem Vektor b. Die Matrix hat insgesamt mn Elemente und b hat m Elemente. Somit gilt für die Eingabegröße $S(A, b) = mn + m$. Bei der Optimisierung kommt noch der Vektor c mit n Elementen hinzu, somit ist die Eingabegröße $S(A, b, c) = mn + m + n$.

Für die ganzen Zahlen ist es sinnvoll die binäre Länge der Elemente als das Hauptmaß für die Eingabegröße zu verwenden, da damit eine Verbindung zu der klassischen Theorie besteht. Die binäre Länge von x (engl. *height*) ist die Anzahl von Bits, die für die Darstellung der Zahl nötig sind, sie ist definiert als $ht(x) = \lceil \log(|x| + 1) \rceil$. Die Eingabegröße von IFP ist $S_{ht}(A, b)$ und ist definiert als $S(A, B)$ multipliziert mit der maximal auftretenden Binärlänge in A und b. Bei IPO wird zusätzlich der Vektor c berücksichtigt.

Die Kosten für das Lösen von LPF und LPO definieren wir als $C(A, b)$ und $C(A, b, c)$ und berechnen als Anzahl der benötigten arithmetischen Operationen für das Lösen des Problems: $\mathcal{A}(A, b)$ und $\mathcal{A}(A, b, c)$. Bei IPF und IPO betrachten wir wieder die binäre Länge. In diesem Fall multiplizieren wir die Anzahl der arithmetischen Operationen mit der größten in der Berechnung auftretenden binären Länge und berechnen somit die Kosten $C_{ht}(A, b)$ bzw. $C_{ht}(A, b, c)$.

Aus der klassischen Theorie ist es bekannt, dass IPF und IPO über die ganzen Zahlen NP-vollständig sind. Für andere Mengen ist es noch nicht bekannt und kann mit Hilfe von der BSS-Modell betrachtet werden.

2.6 Komplexitätstheorie in der Numerik

In der klassischen Theorie sind die Probleme effizient lösbar, wenn sie in polynomieller Zeit gelöst werden können, d.h. für ihre Zeitkomplexität T gilt: $T(x) \leq c \cdot (size(x))^q$, wobei x ist die Eingabe, $size(x)$ ist die Größe der Eingabe und c ist eine Konstante. In der Numerik sind jedoch viele Problem nur mit einer begrenzten Genauigkeit ε lösbar. Je genauer das Ergebnis sein muss, desto länger kann die Berechnung dauern. Deswegen muss der Genauigkeitsparameter bei der Komplexitätsbetrachtung berücksichtigt werden. Für die Genauigkeitswerte $\varepsilon < 1$ sagen wir, dass das Problem effizient lösbar ist falls:

$$T(\varepsilon, x) \leq (|\log(\varepsilon)| + size(x))^q$$

Je kleiner ε desto größer ist $|\log(\varepsilon)|$ und somit die obere Schranke für die effiziente Lösbarkeit. Weiterhin hat bei vielen Problemen die Konditionszahl einen großen Einfluß auf die Laufzeit der Algorithmen. Die Konditionszahl besagt, wie stark die Lösung eines Problems von der Störung ihrer Eingabe abhängig ist. Bei großen Konditionszahlen verändert sogar eine kleine Veränderung der Eingabe das Ergebnis des Algorithmus sehr stark. Für bessere Ergebnisse soll die Laufzeit vergrößert werden. Wir können die Konditionszahl $\mu(x)$ in die obige Bedingung für die effiziente Lösbarkeit folgendermaßen integrieren:

$$T(\varepsilon, x) \leq (|\log(\varepsilon)| + \log(\mu(x)) + size(x))^q$$

Je größer die Konditionszahl $\mu(x)$ ist, desto größer ist $\log(\mu(x))$ und desto länger kann die Berechnung dauern und trotzdem effizient lösbar bleiben.

3 Endlich-dimensionales Modell

In dieser Sektion wird das endlich-dimensionale Modell der BSS-Maschine am Beispiel von der Maschine für das Newton-Verfahren (Abbildung 3.1a) definiert.

Zuerst definieren wir eine *time-T* Haltemenge Ω_T, die alle Startpunkte enthält, für die das Newton-Algorithmus in genau oder weniger als T Iterationen hält:

$$\Omega_T = \{z \in \mathbb{C} \mid |f(N_f^{T'}(z))| < \varepsilon \text{ so dass } 0 \leq T' \leq T\}$$

Wenn wir alle möglichen Anzahlen der Iterationen berücksichtigen, dann erhalten wir die Menge von allen Startpunkten, für die der Algorithmus hält: $\Omega = \bigcup_{0<T<\infty} \Omega_T$. Für diese Startpunkte definieren wir die Input-Output Abbildung $\Phi : \Omega \to \mathbb{C}$, welche die Werte der Eingabe auf das Ergebnis des Algorithmus abbildet. Für einen Startpunkt $z \in \Omega_T$ hat diese Abbildung den Wert der Newton-Iteration nach T' Schritten: $\Phi(z) = N_f^{T'}(z)$. In diesem Fall ist T' die kleinste Anzahl der Iterationen $T > 0$ so dass $z \in \Omega_T$ gilt.

Da die Newton-Maschine den Vergleich $|f(z)| < \varepsilon$ in ihrem Verzweigungsknoten auf den reellen Zahlen macht, ist es sinnvoll auch andere Knoten auf den reellen Zahlen zu definieren. Dafür betrachten wir $\mathbb{C}$ als $\mathbb{R}^2$. Das Newton-Endomorphismus $N_f : \mathbb{C} \to \mathbb{C}$ können wir mit einer folgenden Abbildung ersetzen:

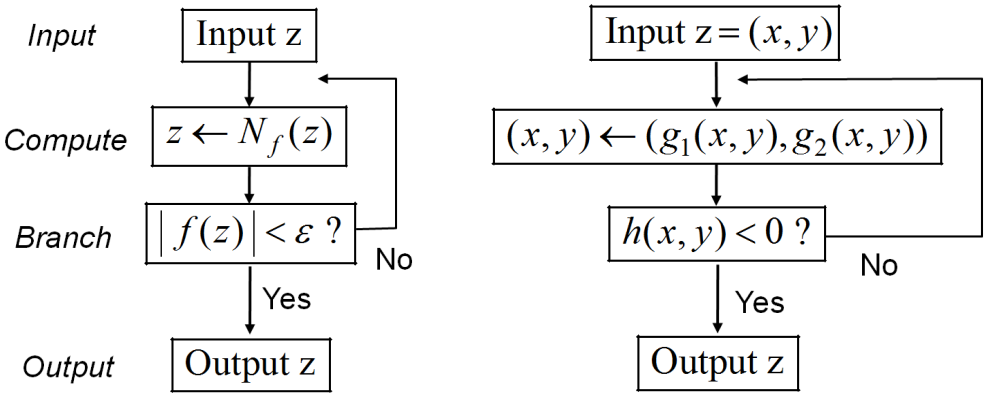
$$g = (g_1, g_2) : \mathbb{R}^2 \to \mathbb{R}^2 \text{ wobei } g_1(x, y) = \text{Re } N_f(x + iy) \text{ und } g_2(x, y) = \text{Im } N_f(x + iy)$$

Die Abbruchbedingung muss entsprechend geändert werden, damit sie Elemente aus $\mathbb{R}^2$ verarbeiten kann. Wir quadrieren die beiden Seiten von der originalen Abbruchbedingung: $|f(z)|^2 < \varepsilon^2$. Daraus folgt:

$$|f(z)|^2 - \varepsilon^2 < 0$$

$$\left(\sqrt{(\text{Re}f)^2 + (\text{Im}f)^2}\right)^2 - \varepsilon^2 < 0$$

$$(\text{Re}f)^2 + (\text{Im}f)^2 - \varepsilon^2 < 0$$



(a) Newton-Maschine mit Knotentypen (b) Endlich-dimensionale Newton-Maschine

Abbildung 3.1: Newton-Maschine mit Knotentypen und ein endlich-dimensionales Modell

Wir definieren $h : \mathbb{R}^2 \to \mathbb{R}$ und $h(x, y) = (\mathrm{Re} f)^2 + (\mathrm{Im} f)^2 - \varepsilon^2$. Somit können wir eine neue Verzweigungsbedingung $h(x, y) < 0$ verwenden. Abbildung 3.1b zeigt die resultierende *endlich-dimensionale* Newton-Maschine.

Wir definieren die allgemeine endlich-dimensionale Maschine M. Sei R ein geordneter kommutativer Ring oder Körper. Die endlich-dimensionale Maschine M über R ist ein gerichteter Graph, wobei gilt:

- $\mathcal{I}_M$: Eingaberaum der Maschine von der Form R^n.
- $\mathcal{S}_M$: Zustandsraum der Maschine von der Form R^m.
- $\mathcal{O}_M$: Ausgaberaum der Maschine von der Form R^l.

In der Abbildung 3.1 sehen wir, dass es vier Arten von Knoten gibt:

1. *Input:* Lineare Abbildung $I : \mathcal{I}_M \to \mathcal{S}_M$. In diesem Knoten wird die Eingabe der Maschine eingelesen. Es gibt nur einen möglichen nächsten Knoten: *Compute*. Der nächste Knoten wird als β_1 bezeichnet.
2. *Compute:* Polynomiale Abbildung (rationale Abbildung falls R ein Körper ist) $g_\eta : \mathcal{S}_M \to \mathcal{S}_M$. In diesem Knoten wird die Berechnung durchgeführt. In dem Fall der Newton-Maschine wird hier der nächste Iterationswert berechnet. Es gibt nur einen möglichen nächsten Knoten: *Branch*. Der nächste Knoten wird als β_η bezeichnet.
3. *Branch:* Polynomiale Abbildung $h_\eta : \mathcal{S}_M \to R$. In diesem Knoten wird das Ergebnis der Berechnung aus dem *Compute*-Knoten evaluiert und eine Entscheidung getroffen, welcher Knoten als nächstes ausgeführt wird. Da R entweder geordnet (z.B. $\mathbb{R}, \mathbb{Z}, \mathbb{Q}$) oder nicht geordnet (z.B. $\mathbb{C}$ in dem Hilbert-Nullstellensatz) sein kann, kann die Verzweigungsbedingung entweder auf einem Ungleichheit- oder auf einem Gleichheitsvergleich basieren. Wir haben zwei mögliche Verzweigungen:
   - $h_\eta(z) \geq 0$ (geordnet) oder $h_\eta(z) = 0$ (nicht geordnet). In diesem Fall wählen wir die *Yes*-Kante in der Abbildung 3.1. Den nächsten ausgeführten Knoten bezeichnen wir als β_η^+.
   - $h_\eta(z) < 0$ (geordnet) oder $h_\eta(z) \neq 0$ (nicht geordnet). In diesem Fall wählen wir die *No*-Kante in der Abbildung 3.1. Den nächsten ausgeführten Knoten bezeichnen wir als β_η^-.
4. *Output:* Lineare Abbildung $O : \mathcal{S}_M \to \mathcal{O}_M$. Beim Erreichen dieses Knotens wird die Maschine halten und der zuletzt berechnete Wert aus dem *Compute*-Knoten wird ausgegeben. Demzufolge gibt es hier keinen Übergang zu einem nächsten Knoten.

Hiermit wurde das endlich-dimensionale Modell der BSS-Maschine und ihre Elemente vorgestellt und definiert. Mit Hilfe von den gezeigten Formalisierungen der verschiedenen Knoten ist es möglich das BSS-Rechenmodell und die Probleme aus der Sektion 2 weiter zu beschreiben und zu analysieren.