Rechnerarithmetik Zusammenfassung

Benedikt Lüken-Winkels

10. April 2019

Inhaltsverzeichnis

1 Grundlagen

1.1 Grundproblem

- Große Zahlen laufen über

1.2 Darstellung Natürlicher Zahlen

Codierung von natürlichen, ganzen Zahlen und rationalen Zahlen

- analog (nicht praktikabel)
- R-adische Notation

⇒ binär in Rechnern

- 8 - 64 Bit Darstellung in Rechnerhardware

Notation endlicher Alphabete Blockcodes:

- passendes k mit $n \leq 2^k$
- Wort wird zerlegt und jeder Block einzeln notiert.
- Bits reichen für die Notation von Zahlen

2 r-adische Zahlen

Mit der r-adischen Notiation kann man jede bel Zahl von $0, ..., r^k - 1$ darstellen. Für ein $k \to \infty$ bedeutet das also jede Zahl ist darstellbar, also $(\cdot)_r$ ist surjektiv.

- Äquivalenz der r-adischen Notationen

⇒ Funktion f ist berechenbar, wenn es eine TM M gibt, mit $f = f_M$

- Ideelle Ebene: Ebene der Zahlen als Konzept
- Reale Ebene: Zahlen als Strings
- TMs agieren auf Strings, nicht auf Zahlen

2.1 Rechenoperationen

Idee für Mehrstellige Operationen Kodieren der Eingabe in einer Kette, wo jeder Teil unabhängig ist

- Aus einem Bitmuster werden k Bitmuster verschiendener Länge pr^k
- In die andere Richtung pr^*
- Das Teilen des Bitmusters in solche Blöcke sorgt dafür, dass die pr_i^k unabhängig voneinander sind.
- Der Zugriff auf das 2. Wort wäre bei der Verwendung eines Trennsymbols in der Komplexität abhängig von der des 1. Wortes

2.1.1 Binäres Zählen

Abschätzung der Laufzeit durch nächst höhere 2er-Potenz.

- Eingabe der Länge n: Naiv: $\log_n$ stellige Binärschreibweise, iterieren über gesamte Eingabe $\Rightarrow O(n \log_n)$
- $k = \log_2(n)$. Aufwandsabschätzung beschränkt durch zählen bis 2^k
  - Es wird immer nur das k-te Zeichen 1. $\log_n$- Teil der Abschätzung wird konstant
- Übertrag kann beliebig weit in die Zahl reinlaufen

2.1.2 Addition

$((\cdot)_r^2, (\cdot)_r)$

- Notation
  - 2 Komponenten müssen zerlegt werden
  - Aufteilen auf 2 Bänder der TM
- Algorithmus = Veralgemeinerung des Schulalgorithmusses
  - Gleiche Länge der Eingaben $\Rightarrow$ Führende 0en einfügen
  - i-ten Übertrag und i-te Komponenten aus Eingabe verrechnen
  - Summe mod r ergibt die i-te Stelle des Ergebnisses
  - Summe div r ergibt den Übertrag (= wie oft passt es rein: 0 wenn w_i kleiner r)
  - Summe $\leq 2r - 1$, also ist Summe div r immer ¡ 2

  $\Rightarrow$ 1 Bit Übertrag reicht
- TM speichert mögliche Ergebnisse Summe in Tabelle (Zustände) $\Rightarrow$ konstante Zeit für jede Berechnung, also **linear**

2.1.3 Subtraktion

Algorithmus

- 2 Komponenten werden subtrahiert und Übertrag addiert
- Übertrag und i-tes Ergebnis wie bei Addition

2.1.4 GMP-Bibliothek, Gnu Multiple Prescision

Zahlen schnell addieren/subtrahieren

- r = 32 (möglichst schnell laden)
- limb := digit
- Addition kann zu Carry-Over führen $\Rightarrow$ muss abgefangen werden
-

2.1.5 Multiplikation

-

2.1.6 Division

-

3 Natürliche und ganze Zahlen