(Master) Berechenbarkeit

v4.0.4.4.3 Text und Gödelnummer

Kategory GmbH & Co. KG
Präsentiert von Jörg Kunze

Beschreibung

Inhalt. Texte, definiert als endliche Folgen von Zeichen aus einem Alphabet, kommen in der Mathematik vor, wenn wir Formeln, Aussagen, Beweise oder Algorithmen mit mathematischen Methoden untersuchen. Ein Satz ist in dem Zusammenhang eine Folge von Zeichen.

Eine Gödelnummer ist ein Code, der aus einem Text oder einem Tupel von natürlichen Zahlen, was eigentlich auch wieder eine Art Text ist, eine natürliche Zahl macht. Umkehrbar. Damit können wir Funktionen, die auf Texten oder auf Tupeln arbeiten, auf Funktionen von $\mathbb{N}$ nach $\mathbb{N}$ zurückführen.

Unsere ganze Berechenbarkeitstheorie führen wir auf $\mathbb{N}$ durch. Wir haben aber immer im Kopf, dass alles, was wir sagen, auch für Funktionen auf Texten gilt.

Beispiele für Textfunktionen in der mathematischen Logik sind: Ist ein Ausdruck syntaktisch korrekt. Ist ein Satz beweisbar. Ist eine Folge von Aussagen ein Beweis. Das Einsetzen eines Ausdrucks für eine Variable. Ermitteln, ob ein Algorithmus gegeben durch einen formalen Programm-Code in eine Endlos-Schleife gerät.

Wir hätten auch Texte als das ursprüngliche ansehen können und dann Zahlen als Wörter auffassen können, wie z. B. durch die Dezimaldarstellung. Tatsächlich kommt dies dem tatsächlichen Geschehen bei der mathematischen Arbeit näher.

Eine Gödelnummer ist eine Funktion von der Ausgangsmenge in $\mathbb{N}$ mit bestimmten Eigenschaften. Die Bilder dieses Funktion heißen auch Codes. Die Ausgangsmenge ist z. B. die Menge der Wörter über einem Alphabet oder die Menge der Tupel über $\mathbb{N}$. Die Eigenschaften sind:

- Injektiv
- Berechenbar
- Die Frage, ob eine Zahl im Bild ist, ist berechenbar
- Die auf dem Bild definierte Umkehrfunktion ist ebenfalls berechenbar.

Mit anderen Worten können wir zwischen Wörtern/Tupeln und deren Codes verlustfrei mit Hilfe eines Computerprogramms hin und her rechnen.

Präsentiert. Von Jörg Kunze

Voraussetzungen. Mu-rekursive Funktionen, berechenbare Funktionen, Turing-Maschinen

Text. Der Begleittext als PDF und als LaTeX findet sich unter `https://github.com/kategory/kategoryMathematik/tree/main/v4%20Master/v4.0%20Berechenbarkeit/v4.0.4.4%20Church-Markov-Turing%20These`

Meine Videos. Siehe auch in den folgenden Videos:

v4.0.3 (Master) Berechenbarkeit - Turing Maschine
`https://youtu.be/29UcyRumdFo`

v4.0.4 (Master) Berechenbarkeit - Berechenbare Funktionen
`https://youtu.be/tARmHFIP32o`

v4.0.4.2 (Master) Berechenbarkeit - Rekursive Funktionen
`https://youtu.be/tFEn2GoLLEQ`

Quellen. Siehe auch in den folgenden Seiten:
`https://de.wikipedia.org/wiki/Church-Turing-These`
`https://de.wikipedia.org/wiki/Turing-Vollst%C3%A4ndigkeit`
`https://en.wikipedia.org/wiki/Turing_completeness`

Buch. Grundlage ist folgendes Buch:
Computability
A Mathematical Sketchbook
Douglas S. Bridges

Springer-Verlag New York Inc. 2013
978-1-4612-6925-0 (ISBN)

Das Video. Das Video hierzu ist zu finden unter `hhh`

1. Church-Markov-Turing These

1.1. **Drei Definitionen von berechenbar.** Es gibt drei übliche Definitionen von berechenbaren Funktionen:
Turing-Maschinen
Lambda-Kalkül
Mu-rekursiv

1.2. **Turing-äquivalent.** Eine System oder ein Begriff, welches Funktionen liefert oder definiert, ist **Turing-vollständig**, wenn jede Turing-Maschinen-berechenbare Funktion geliefert wird. Die können wir z. B. dadurch beweisen, dass in diesem System ein Turing-Maschinen-Simulator implementiert werden kann.

Können alle Funktionen eines Turing-vollständigen Systems auch von einer Turing-Maschine geliefert werden, heißt das System **Turing-äquivalent**. Das können wir z. B. dadurch beweisen, dass wir das System in einer Turing-Maschine implementieren.

1.3. **Church-Markov-Turing These.** Die Church-Markov-Turing These, die weder mathematischer Satz noch Vermutung ist, besagt, dass jede auch nur erdenkliche halbwegs vernünftige Definition des Berechenbarkeits-Begriffs Turing-äquivalent ist. Die These heißt auch Church-Turing These oder nur Church These.

Formal bewiesen ist es als mathematischer Satz für Turing-Maschinen, Lambda-Kalkül, Mu-rekursiv, und alle normalen Programmiersprachen, wie C, C++, Python, JavaScript, Lisp, Prolog, Java, Haskell ... Bei den Programmiersprachen nehmen wir dann an, dass sie über einen unendlichen Speicher verfügen. Alle Programmiersprachen sind prinzipiell gleich. Wir können den Raum der erreichbaren Funktionen nicht erweitern.

Ein konzeptioneller Unterschied von JavaScript und einer Turing-Maschine ist, dass letztere einen unendlichen Speicher hat. Hier ist allerdings zu bedenken, dass die Turing-Maschine zu keinem Zeitpunkt unendliche viel Speicher nutzt. Es sind immer nur endliche viele Speicherplätze mit einer 1 belegt. Der Rest ist 0. Dies entspricht en wenig der Idee des potentiell unendlichen. JavaScript und andere real-technische Programmiersprachen sind in folgenden Sinne pragmatisch potentiell unendlich: wenn eine Berechnung wegen zu wenig Speicher abbricht, können wir neuen RAM oder Plattenspeicher kaufen, und das ganze dann nochmal versuchen. Oder wir warten auf die nächste Rechner-Generation. Ein Schritt, den die Menschheit immer wieder macht.

1.4. **JavaScript.** JavaScript ist Turing-äquivalent, wenn wir von der Speicherbegrenzung absehen. In der Spezifikation von JavaScript wird nicht gesagt, dass nur Speicher bis zu einer bestimmten Größe erlaubt ist. Über das Adressierungsmodel oder Pointer-Datentypen wird keine Aussage gemacht.

Ein realer Computer aber, der isoliert dasteht also keinen Zugriff auf (eventuell mit der Zeit wachsende) externe (Cloud-) Speicher hat ist auch mit JavaScript nicht Turing-äquivalent noch nicht einmal Turing-vollständig.

Ein Augenmerk muss dabei auf Zahlen gerichtet werden. Normale Zahlen in JavaScript sind vom Typ Number, eine 64-bit floating point IEEE 754 Darstellung. Der höchste Wert hier ist
`Number.MAX_VALUE` $= 2^{1024} - 2^{971} \approx 1.79769313486231571^{308}$
Allerdings sind hier Dezimalstellen abgeschnitten. Unabgeschnitten geht es nur bis
`Number.MAX_SAFE_INTEGER` $2^{53} - 1 = 9007199254740991$.

Eine Lösung ist, mit BigInt zu arbeiten. Dies ist ein Datentyp, der beliebig große ganze Zahlen mit all ihren Dezimalstellen abbilden kann. In der Spezifikation findet sich, meines Wissens, keine Obergrenze. Der zur Verfügung stehende Speicher liefert natürlich eine solche. BigInt können über den Konstruktor oder auch über Literale (ganze Zahlen mit einem n am Ende) erzeugt werden:

```
x = BigInt( 42 );
y = 4875498574329574398572940385790234857423n;
```

1.5. **Mathematische Grundlagen.** Die Auswirkung auf die mathematischen Grundlagen und Logik: ein Algorithmus, wie der euklidische, sollte Turing-maschinen-berechenbar sein, um als solcher zu gelten. Definitionen von „gültiger Term“, „wohl geformte mathematische Aussage“ oder „korrekter Beweis“ sollten durch Turing-maschinen-berechenbare Funktionen auf den Zeichenketten von Term, Aussage oder Beweis definiert werden können. Sie sollten also in JavaScript programmiert werden können.

$$x + \text{text} \tag{1}$$

Literatur

[Douglas2013] Douglas S. Bridges, *Computability, A Mathematical Sketchbook*, Springer, Berlin Heidelberg New York 2013, ISBN 978-1-4612-6925-0 (ISBN).

Symbolverzeichnis

$\mathbb{N}$	Die Menge der natürlichen Zahlen (mit Null): $\{0, 1, 2, 3, \cdots\}$
n, k, l, s, i, x_i, q	Natürliche Zahlen
$A(k, n)$	Ackermannfunktion
f, g, h	Funktionen
$s()$	Nachfolgerfunktion: $s(n) := n + 1$