

(Master) Berechenbarkeit

v4.0.4.4 Church-Markov-Turing These

Kategory GmbH & Co. KG

Präsentiert von Jörg Kunze

Copyright (C) 2023 Kategory GmbH & Co. KG

BESCHREIBUNG

Inhalt. Die Church-Markov-Turing These besagt, dass jede denkbare Formalisierung des Begriffs der Berechenbarkeit zur selben Menge berechenbarer Funktion führt. Anders gesagt, die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein. Sie besagt unter anderem auch, dass wir mit jeder erdenklichen Programmiersprache die selben Programme schreiben können.

Es ist eine These. Kein Theorem, da es nicht bewiesen ist. Noch nicht einmal eine mathematische Aussage, sodass es auch nie bewiesen werden kann. Es ist also auch keine Vermutung. Aber alles, was wir bisher in dieser Richtung unternommen haben, stützt die Church-Markov-Turing These.

Ein System, welches zu den selben Funktionen führt, wie die Turing-Maschinen heißt Turing-äquivalent.

Mathematisch beweisen können wir Teile dieser These. So z. B. die Äquivalenz von rekursiven, Lambda-Kalkül und Turing-Maschinen Funktionen. Wir können ebenfalls beweisen, dass die üblichen Programmiersprachen Turing-äquivalent sind. Wenn wir also zu einem Problem eine JavaScript Implementierung im Kopf haben, können wir davon ausgehen, dass es Turing-maschinen-berechenbar ist.

Die Turing-Äquivalenz von z. B. JavaScript könnte dadurch bewiesen werden, dass wir eine Turing-Maschine in JavaScript simulieren und dass wir einen JavaScript-Interpreter in einer Turing-Maschine implementieren.

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 ein wenig der Idee des potentiell unendlichen.

JavaScript und andere real-technische Programmiersprachen sind in folgendem 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.

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.

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.%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)

Lizenz. Dieser Text und das Video sind freie Software. Sie können es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation veröffentlicht, weitergeben und/oder modifizieren, entweder gemäß Version 3 der Lizenz oder (nach Ihrer Option) jeder späteren Version.

Die Veröffentlichung von Text und Video erfolgt in der Hoffnung, dass es Ihnen von Nutzen sein wird, aber OHNE IRGENDNEINE GARANTIE, sogar ohne die implizite Garantie der MARKTREIFE oder der VERWENDBARKEIT FÜR EINEN BESTIMMTEN ZWECK. Details finden Sie in der GNU General Public License.

Sie sollten ein Exemplar der GNU General Public License zusammen mit diesem Text erhalten haben (zu finden im selben Git-Projekt). Falls nicht, siehe <http://www.gnu.org/licenses/>.

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 Adressierungsmodell 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

$$\text{Number.MAX_VALUE} = 2^{1024} - 2^{971} \approx 1.797693134862315710^{308}$$

Allerdings sind hier Dezimalstellen abgeschnitten. Unabgeschnitten geht es nur bis

$$\text{Number.MAX_SAFE_INTEGER} 2^{53} - 1 = 9007199254740991.$$

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.

(1) $x + \text{text}$

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): $\{0, 1, 2, 3, \dots\}$
n, k, l, s, i, x_i, q	Natürliche Zahlen
$A(k, n)$	Ackermannfunktion
f, g, h	Funktionen
$s()$	Nachfolgerfunktion: $s(n) := n + 1$