12 Turingmaschinen

12.1 das hier hatten wir ja schon

Die Anfänge der Turingmaschinen solten ja schon im zwölften Tutorium behandelt worden sein. Daher hier nur der neue Stoff:

12.2 Berechnungskomplexität

- Noch mal der Hinweis aus dem Skript: Der Einfachheit halber wollen wir in diesem Abschnitt davon ausgehen, dass wir ausschließlich mit Turingmaschinen zu tun haben, die für jede Eingabe halten.

  In nächsten Abschnitt werden wir dann aber wieder gerade von dem allgemeinen Fall ausgehen, dass eine Turingmaschine für manche Eingaben *nicht* hält.

- Das ist vielleicht etwas verwirrend für die Studenten. Aber der Formalismus für Komplexitätstheorie, bei dem alles für manchmal nicht haltende TM durchgezogen wird, ist auch nicht gerade so toll. Und man muss aufpassen.

- Lieber die Studenten anflehen, sie mögen doch bitte glauben, dass die Annahme des Immer-Haltens ok ist. Wir werden auch in der Vorlesung was dazu sagen.

12.2.1 Komplexitätsmaße

- bei Komplexitätsmaßen üblich: z. B. bei der Zeitkomplexität Angabe des schlimmsten Falles in Abhängigkeit von der Eingabegröße (und nicht für jede Eingabe einzeln).

- „Auflösen“ der Rekursion $\mathrm{Time}(n+2) \leq 2n+1+\mathrm{Time}(n-2)$ und Ergebnis $\mathrm{Time}(n) \in \mathrm{O}\left(n^2\right)$ ggf. klar machen können.

- Bitte Zusammenhänge zwischen Zeit- und Raumkomplexität klar machen.

- Um zu sehen, dass man auf linearem Platz exponentielle Zeit verbraten kann:

  - man baue noch eine TM: auf dem Band steht anfangs eine Folge von Nullen. Aufgabe der TM: Solange auf dem Band nicht eine Folge nur aus Einsen steht, immer wieder die TM von weiter vorne anwenden, die die Zahl um 1 erhöht.
  - Wenn anfangs n Nullen auf dem Band stehen, dann wird 2^n-1 mal die 1. TM ausgeführt; das macht insgesamt offensichtlich $\geq 2^n$ Schritte.

  - Für die, die es genauer machen wollen: **Achtung:** $\Theta\left((2^n-1)n\right) \not\subseteq \mathrm{O}\left(2^n\right)$, aber natürlich z. B. $\Theta\left((2^n-1)n\right) \subseteq \mathrm{O}\left(3^n\right)$.

- **GANZ WICHTIG:** *Niemals* von der Zeitkomplexität o. ä. eines Problems reden.

  - Algorithmen haben eine Laufzeit. Probleme nicht.
  - **Achtung:** Der Versuch die Laufzeit eines Problems als die der schnellsten Algorithmen zur Lösung des Problems zu definieren funktioniert nicht.

    Es gibt Probleme, für die es keine schnellsten Algorithmen gibt. Sondern zu jedem Algorithmus für ein solches Problem gibt es einen anderen, der *um mehr als einen konstanten Faktor (!)* schneller ist.

12.2.2 Komplexitätsklassen

- Aus dem Skript: Wichtig:

  - Eine Komplexitätsklasse ist eine Menge von Problemen — und *nicht* von Algorithmen.
  - Wir beschränken uns im folgenden wieder auf Entscheidungsprobleme.

12.3 Unentscheidbare Probleme

- Als erstes noch mal den prinzipiellen Unterschied zwischen „nur in Zeit $2^{2^{2^n}}$ berechenbar“ und „überhaupt nicht berechenbar“ klar machen

- auch wenn man in der Praxis nie $2^{2^{2^n}}$ Zeit hat.

12.3.1 Codierungen von Turingmaschinen

- Die im Skript beschriebene Codierung ist so gewählt, weil

  - man leicht von einem Wort überprüfen kann, ob es eine TM codiert, und
  - sie bequem ist, um eine TM zu simulieren, wenn man ihre Codierung hat.

- Wer mag, kann ja eine kleine TM codieren.

- Da wir bei der Unentscheidbarkeit des Halteproblems ohne universelle TM auskomme, werden wir das Thema “universelle TM” nur kurz ansprechen.

12.3.2 Das Halteproblem

Diagonalisierung:

- wird nur relativ allgemein beschreiben, aber eben nur den Kern.
- Der Kern sollte klar sein: Die „verdorbene Diagonale“ $\overline{d}$ (so nennen wir das immer; wenn Sie einen besseren Begriff haben ...) unterscheidet sich von jeder Zeile der Tabelle.
- Die Art der Ausnutzung der Idee variiert.
  - Manchmal weiß man, dass die Zeilen *alle* Möglichkeiten einer gewissen Art umfassen, dann ist also $\overline{d}$ sicher nicht von der gewissen Art; es gibt also etwas, was nicht von der gewissen Art ist. (z. B. Komplexitätstheorie: es gibt ein Problem, das nicht in Zeit n^2 o.ä. lösbar ist) (aber wie sich zeigt z. B. in Zeit $n^{2+\varepsilon}$, wenn man die Diagonale vorsichtig genug verdirbt).
  - Manchmal, z. B. bei der Überabzählbarkeit von $\mathbb{R}$, ist $\overline{d}$ Zeuge dafür, dass die Tabelle nie vollständig sein kann.
  - beim Halteproblem ist es noch ein bisschen anders.

Halteproblem:

- Die Aussage (und der Beweis) sollten sitzen.
- Statt konkret über TM reden wir nur noch von Algorithmen.

12.3.3 Die Busy-Beaver-Funktion

- Aus Zeitgründen müssen wir uns hier aufs Staunen beschränken. Vielleicht macht das ja den ein oder anderen neugierig.
- Wenn ich mich recht erinnere, hat der Wertebereich von bb() übrigens die Eigenschaft, dass weder er noch sein Komplement aufzählbar ist. Also noch schlimmer als H ...

13 Äquivalenzrelationen

13.1 Definition

- die Eigenschaften reflexiv, symmetrisch und transitiv an Beispielrelationen klar machen

- evtl. auch Relationen vorführen, die nur zwei oder eine oder gar keine dieser Eigenschaften haben
- Darstellung von Relationen als gerichtete Graphen: Woran sieht man
  - Reflexivität?
  - Symmetrie?
  - Transitivität?
- Wie sieht der Graph einer Äquivalenzrelation aus: „Klumpen“, in denen jeder mit jedem verbunden ist, zwischen den Klumpen nichts (die Klumpen heißen später Äquivalenzklassen)

13.2 Äquivalenzrelationen von Nerode

- die Definition der Nerode-Äquivalenz verstand jedenfalls ich nicht auf Anhieb
- Manche brauchen vielleicht immer noch Anleitung, die Definition überhaupt richtig zu lesen.
- vielleicht hilft es, auch das zu diskutieren:
  - man nehme ein L, das von einem endlichen Akzeptor erkannt wird
  - man nehme zwei Wörter w_1, w_2 die *nicht* $\equiv_L$-äquivalent sind
  - Was kann man über $f^*(z_0, w_1)$ und $f^*(z_0, w_2)$ sagen? Sie müssen verschieden sein, denn sonst $f^*(z_0, w_1) = f^*(z_0, w_2)$ und dann auch für jedes Suffix w: $f^*(z_0, w_1 w) = f^*(z_0, w_2 w)$, also werden für jedes Suffix entweder beide Wörter $w_1 w$ und $w_2 w$ oder keines akzeptiert, und dann wären w_1 und w_2 ja äquivalent.

13.3 Äquivalenzklassen und Faktormengen

noch mal das Beispiel Kongruenz modulo n; nehmen wir $n = 5$; also $x \equiv y \pmod 5$; das gilt, wenn $x - y$ ganzzahliges Vielfaches von 5 ist:

- $\ldots, -10, -5, 0, 5, 10, \ldots$ sind alle äquivalent zueinander, also
  $[0] = \{\ldots, -10, -5, 0, 5, 10, \ldots\}$ oder kurz $[0] = 5\mathbb{Z}$ (mit der Komplexschreibweise aus Abschnitt 13.2.4 im Skript)
  statt [0] hätte man auch [5] oder [−10] oder [2783012931025] schreiben können.
- da $1 \not\equiv 0 \pmod 5$, ist [1] eine *andere* Äquivalenzklasse.
  $[1] = 1 + 5\mathbb{Z}$; genauso gut könnte man schreiben $[1] = -24 + 5\mathbb{Z}$

- Bitte klar machen: für $x \neq y$ kann $[x] = [y]$ sein
- Beweisen: wenn $x \equiv y$, dann $[x] = [y]$
  - wenn $z \in [x]$, dann $x \equiv z$, also wegen Symm. auch $z \equiv x$
  - mit $x \equiv y$ und Transitivität folgt $z \equiv y$,
  - also $y \equiv z$, also $z \in [y]$
  - also $[x] \subseteq [y]$.
  - umgekehrt geht es genauso.
- Beweisen: Wenn ein z sowohl in $[x]$ als auch in $[y]$ ist, dann ist $[x] = [y]$.
  - Wenn $z \in [x]$ und $z \in [y]$, dann $x \equiv z$ und $y \equiv z$,
  - also wegen Symmetrie $x \equiv z$ und $z \equiv y$,
  - also wegen Transitivität $x \equiv y$
  - also (eben gesehen) $[x] = [y]$
  - Äquivalenzklassen sind also entweder disjunkt oder gleich. „halbe Überlappungen“ gibt es nicht

Faktormenge von $\mathbb{Z}$ für Kongruenz modulo 5

- hinreichend langes Überlegen zeigt: die Äquivalenzklassen [0], [1], [2], [3] und [4] sind alle paarweise verschieden: für je zwei der Zahlen ist die Differenz offensichtlich positiv, aber echt kleiner als 5.
- Aber für jedes andere $x \in \mathbb{Z}$ gibt es eine äquivalente Zahl zwischen 0 und 4, nämlich den Rest bei Division durch 5.
- Also gibt es nur fünf Äquivalenzklassen:

$$\mathbb{Z}_{/\equiv_5} = \mathbb{Z}_5 = \{[0], [1], [2], [3], [4]\}$$

Faktormenge für Nerode-Äquivalenz

- Machen Sie sich bitte die Äquivalenzklassen von $\equiv_L$ aus den Skriptbeispielen klar, so dass Sie sie erklären können.