16 TURINGMASCHINEN

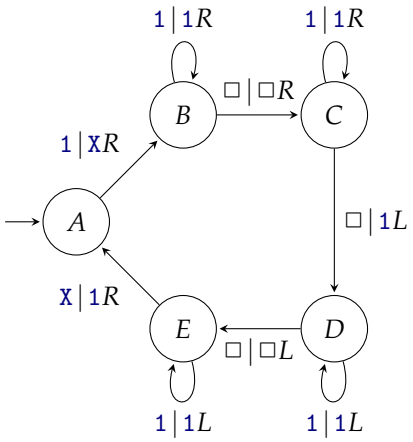
partielle Funktionen

- ggf. noch mal kurz drauf eingehen
- „wie eine normale Funktion, aber an manchen Stellen evtl. nicht definiert“

16.1 ALAN MATHISON TURING

16.2 TURINGMASCHINEN

- wir betrachten nur die simpelste Variante:
  - nur ein Kopf
  - nur ein Arbeitsband
  - keine separaten „Spezialbänder“ für Eingaben oder Ausgaben
- habe die Arbeitsweise auf 3 Funktionen f, g, m aufgeteilt, weil man da einfacher hinschreiben kann, was ein Schritt ist
  1. Feld mit nächstem Symbol beschriften
  2. in neuen Zustand übergehen
  3. Kopf bewegen
- weiteres Beispiel:



	A	B	C	D	E
□		□, R, C	1, L, D	□, L, E	
1	X, R, B	1, R, B	1, R, C	1, L, D	1, L, E
X					1, R, A

Wenn ich mich nicht vertan habe, kopiert diese TM ein Wort 1^k auf einem leeren Band, so dass hinterher $\cdots \square 1^k \square 1^k \square \cdots$ da steht, falls man auf der ersten 1 startet. Richtig?

- Verallgemeinerung fürs Kopieren von Wörtern über $\{0,1\}$: Man muss sich auf dem Weg nach rechts merken, was mit `X` überschrieben wurde, und man muss auf dem Weg nach rechts und zurück sowohl `1` als auch `0` überlaufen.

16.2.1 Berechnungen

- Wichtig: Darauf eingehen, dass es unendliche Berechnungen gibt — bei TM genauso wie in Java.

16.2.2 Eingaben für Turingmaschinen

16.2.3 Ergebnisse von Turingmaschinen

- die Palindrommaschine sollten Sie sich klar gemacht haben
- ich beschränke mich weitgehend auf Akzeptoren, aber wenn Sie wollen, können Sie eine TM besprechen, die eine binär dargestellte Zahl um 1 erhöht:

	r	c_0	c_1	h
`0`	$0, R, r$	$0, L, c_0$	$1, L, c_0$	
`1`	$1, R, r$	$1, L, c_0$	$0, L, c_1$	
$\square$	$\square, L, c_1$	$\square, R, h$	$1, L, c_0$	

  Auf diese Maschine komme ich später noch mal zurück.

16.3 Berechnungskomplexität

- Noch mal der Hinweis aus dem Skript: Der Einfachheit halber wollen wir in diesem Abschnitt und im nachfolgenden Abschnitt 16.4 davon ausgehen, dass wir ausschließlich mit Turingmaschinen zu tun haben, die für jede Eingabe halten.

  In Abschnitt 16.5 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. Ich werde auch in der Vorlesung was dazu sagen.

16.3.1 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 $\text{Time}(n+2) \leq 2n+1+\text{Time}(n-2)$ und Ergebnis $\text{Time}(n) \in \text{O}\left(n^2\right)$ ggf. klar machen können.
- Palindromerkennung ist übrigens einer der schönen Fälle, in denen man beweisen kann, dass jede 1-Kopf-TM Laufzeit in $\Omega\left(n^2\right)$ hat.

- Bitte machen Sie sich die Zusammenhänge zwischen Zeit- und Raumkomplexität klar.
- 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:** $\Theta\left((2^n-1)n\right) \not\subseteq \text{O}\left(2^n\right)$, aber natürlich z. B. $\Theta\left((2^n-1)n\right) \subseteq \text{O}\left(3^n\right)$.

- **GANZ WICHTIG:** Reden Sie *niemals* von der Zeitkomplexität eines Problems.
  - 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.

16.3.2 Komplexitätsklassen

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

16.4 UNENTSCHEIDBARE PROBLEME

- Als erstes noch mal den prizipiellen 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.

16.4.1 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.

- Ich habe faulerweise darauf verzichtet, auch noch zu sagen, wie man bei TM-Akzeptoren die akzeptierenden Zustände codiert. Man denke sich was bequemes aus.
- Wer mag, kann ja eine kleine TM codieren.

- Da ich bei der Unentscheidbarkeit des Halteproblems ohne universelle TM auskomme, habe ich das Thema nur kurz angesprochen.
- Falls ich viel Zeit habe, liefere ich eine Beschreibung einer universellen TM. Ansonsten:
- Wer Lust hat, kann sich ja mal für die beschriebene Codierung eine universelle TM überlegen. Technisch braucht man nicht viel.
- Falls im Tutorium Zeit ist oder entsprechende Fragen kommen, kann man die universelle TM ja mal „auf hohem Niveau“ beschreiben.

16.4.2 Das Halteproblem

Diagonalisierung:

- habe ich relativ allgemein beschreiben, aber eben nur den Kern.
- Der Kern sollte gaaaanz klar sein: Die „verdorbene Diagonale“ $\overline{d}$ (so nenn ich das immer; wenn Sie einen besseren Begriff haben dots 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, dass 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 müssen sitzen.
- Statt konkret über TM rede ich nur noch von Algorithmen. Man mache sich im Zweifelsfall klar, wie eine TM das jeweils bewerkstelligen könnte. Ich wollte hier aber keinen Workshop über „TM-Tools“ und „TM-Libraries“ veranstalten.
- Ich diskutiere nicht die Tatsache, dass H als formale Sprache immerhin erkennbar (also wie man auch sagt aufzählbar) ist. Dafür bräuchte man universelle TM.

16.4.3 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 $\mathrm{bb}()$ übrigens die Eigenschaft, dass weder er noch sein Komplement aufzählbar ist. Also noch schlimmer als H ...

