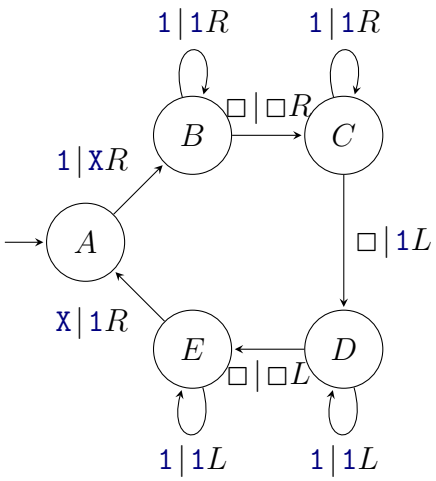
12 Turingmaschinen

partielle Funktionen

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

12.1 Bestandteile einer Turingmaschine

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



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

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

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

12.1.1 Berechnungen

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

12.1.2 Ergebnisse von Turingmaschinen

- die Palindrommaschine aus der Vorlesung (ca Folie 32/33) sollte klar sein
- wir beschränken uns weitgehend auf Akzeptoren, aber wer will, kann eine TM besprechen, die eine binär dargestellte Zahl um 1 erhöht:

	r	c_0	c_1	h
$\mathtt{0}$	$r, \mathtt{0}, R$	$c_0, \mathtt{0}, L$	$c_0, \mathtt{1}, L$	
$\mathtt{1}$	$r, \mathtt{1}, R$	$c_0, \mathtt{1}, L$	$c_1, \mathtt{0}, L$	
$\square$	$c_1, \square, L$	$h, \square, R$	$c_0, \mathtt{1}, L$	

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 $\text{Time}(n+2) \leq 2n+1+\text{Time}(n-2)$ und Ergebnis $\text{Time}(n) \in \text{O}\,(n^2)$ ggf. klar machen können.
- Palindromerkennung ist übrigens einer der schönen Fälle, in denen man beweisen kann, dass jede 1-Kopf-TM Laufzeit in $\Omega\,(n^2)$ hat.
- 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:** $\Theta\,((2^n-1)n) \not\subseteq \text{O}\,(2^n)$, aber natürlich z. B. $\Theta\,((2^n-1)n) \subseteq \text{O}\,(3^n)$.
- **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.