UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Turing Maschinen

Ivo Blöchliger / Ulrich Ultes-Nitsche

Departement für Informatik, Universität Freiburg

Lehrerweiterbildung *Turing Maschinen — mal ganz praktisch*

Die Turing Maschine

- Vorgeschlagen von Alan Turing im Jahr 1936
- Mathematisches Modell zum Studium der Berechenbarkeit
- Kein direktes praktisches Interesse
- Universelle "Rechen" Machine

Motivation

- Mehrfach im EFI eingesetzt
- Die Programmierung einer TM ist herausfordernd
- Illustration der Codierung (assembler-like)
- Möglicher Startpunkt für reguläre Ausdrücke

Ziel des heutigen Tages

- Eine universelle TM implementieren
- Schlüsselalgorithmen mit einer TM programmieren
- Sich über die Codiermöglichkeiten austauschen
- Theoretisch und praktisch relevante Resultate sehen

Bestandteile

- Ein unendliches Band, das in Zellen unterteilt ist
- Ein endliches Alphabet, das u.a. das “Blank”-Symbol enthält
- Ein Schreib-/Lesekopf
- Ein Mechanismus vor- oder zurückzugehen
- Eine endliche Anzahl von Zuständen
- Eine Übergangstabelle

Übergangstabelle $\leftrightarrow$ Maschinencode

Alphabet: $\{., 1\}$
Band: $1\ 1\ 1 \ldots 1\ .\ .\ .\ .$

Zustand	Lesen	Schreiben	Bewegung	Nachfolgezustand
e_1	.	1	L	e_2
	1	1	R	e_1
e_2	1	1	L	e_2
	.	.	R	**stop**

“Assembler” Code

```
#tape 111

e1 R
    . 1 L e2

e2 L
    . . R stop
```

```
[    0] start
. . . . . . . . . .[1]1 1 . . . . . . .
[    1] e1
. . . . . . . . . . 1[1]1 . . . . . . . .
[    2] e1
. . . . . . . . . . 1 1[1]. . . . . . . . .
[    3] e1
. . . . . . . . 1 1 1[.]. . . . . . . . .
[    4] e1
. . . . . . . . . 1 1[1]1 . . . . . . . .
[    5] e2
. . . . . . . . . . 1[1]1 1 . . . . . . .
[    6] e2
. . . . . . . . . . .[1]1 1 1 . . . . . .
[    7] e2
. . . . . . . . . . .[.]1 1 1 1 . . . . .
[    8] e2
. . . . . . . . . . .[1]1 1 1 . . . . . .
```

Codestruktur

- Inspiriert vom Originalartikel von Turing von 1936
- Ausdrucksstarke Notation
- “Komfortable” Programmierung einer TM
- Möglichkeit, *m*-Funktionen zu schreiben

Codestruktur

- Kommentare beginnen mit #
- Spezialkommentar beginnt mit `#tape`
- Definition des Alphabets (automatisch)
- Definition der Zustände
- Definition der *m*-Funktionen

Zustandsdefinition

- Header
  Name [Richtung per default [Zustand per default]]
- Definitionszeilen
  - einfache Definition
    read write direction nextState
  - Definition von Kommandosequenzen
    read {commands} nextState
    - Kommandos: L, R oder Px, worin x ein Symbol ist, das geschrieben wird
- Leerzeile, um Zustände zu separieren!

Beispiele von Zustandsdefinitionen

```
seq
  . {P1 R P. R P1 L L} bla
  1 . R seq

bla L seq
  1 {L L} bla
```

seq: Für Symbole ausser . und 1 ändert sich nichts.
bla: Für Symbole ausser 1 gehe nach links und wechsele in Zustand seq. Ansonsten gehe zweimal nach links und bleibe im Zustand bla.

```
foo
   * 1 R foo
   . . L bar

bar L
  [01X] * R foo
```

foo: Für Symbole ausser . schreibe 1 und gehe nach rechts.
bar: Für 0,1 oder X schreibe das selbe Symbol, gehe nach rechts, wechsele in Zustand foo. Sonst gehe nach links und bleibe in bar.

* und [...]

- *: Als gelesenes Symbol: alle Symbole des Alphabets
- [...]: Wie *, aber beschränkt auf die Symbole zwischen [und]

  Bemerkung: [...] darf nicht @1 etc. enthalten
- *: Als geschriebenes Symbol oder Argument einer *m*-Funktion: Symbol, das gelesen wurde.

m-Funktionen

- Header: `@`*name*`(`*x* `;` *y*`)`
  worin *x* die Anzahl der zu übergebenden Symbole als Argumente und *y* die Anzahl der zu übergebenden Zustände ist.
- Body: Menge von lokalen Zustandsdefinitionen.
  Bemerkung: ein lokaler Zustand verdeckt einen gleichnamigen globalen Zustand
- Footer: `@end`

Beispiel einer *m*-Funktion

```
# dummy state
# does nothing but transit
start
  * * N @findSymbol(1 0; stop)

# searches for the 1st
# given symbol on the
# right, replaces it by the
# 2nd symbol and then transit
# into the given state
@findSymbol(2;1)
  find
    @1 @2 N $1
@end
```

@1, @2, etc. sind die Symbole, die als Argument übergeben werden.
Beim Aufruf werden die Symbole im Argument durch Leerzeichen separiert und können * oder wieder @1, @2, etc. enthalten

$1 $2, etc. sind die übergebenen Zustände, wiederum beim Aufruf durch Leerzeichen separiert. Die Argument können selbst wieder Aufrufe von *m*-Funktionen sein.

Spezielle und verbotene Symbole

- Leerzeichen (Trennsymbol)
- . Blank
- * (wildcard)
- @ (Symbolvariable in *m*-Funktion oder *m*-Aufruf)
- $ (Zustandsvariable)
- [und] (Symbolbereich)
- _ (underscore, wird intern verwendet)
- # Beginn eines Kommentars

Richtungen

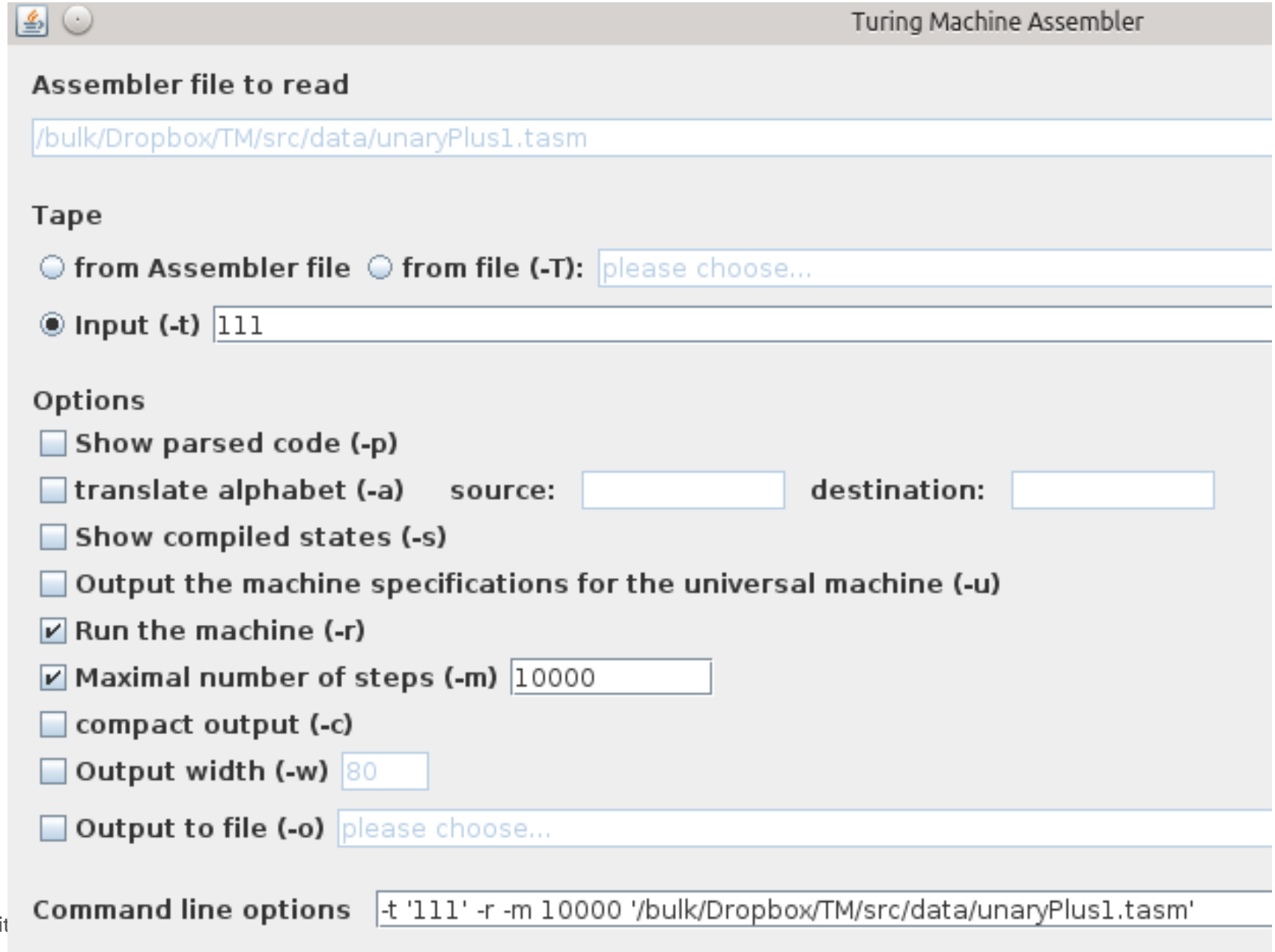
- R, L oder N (keine Bewegung)

N ist nicht notwendig, macht aber die Programmierung einfacher.

Verwendung des Programms

- Java7 oder Java8: TM.jar
- Kommandozeile
- Mögliche Ausgabe in eine Datei
- Eingabe: Datei mit Turing-Assembler

Verwendung des Programms



Übung 1: Unär mal Zwei

- Input: unäre Zahl, z.B. 11111 (5).
  Schreib-/Lesekopf auf der ersten 1 (von links).
- Ausgabe: die doppelte unäre Zahl.
  Schreib-/Lesekopf auf der ersten 1 (von links).

Strategien

- Zusätzliche Symbole, um die verarbeiteten Positionen zu markieren.
- Verwendung nur jeder zweiten Zelle, um die Zahl zu speichern, die anderen Zellen werden zur Markierung verwendet.

- Skizziert die Operationen und/oder kommentiert die Zustände

Übung 2: Binäre plus 1

- Eingabe: Binärzahl, little-endian 111001 (39).
  Schreib-/Lesekopf auf der ersten Ziffer (von links).
- Ausgabe: um 1 erhöhte Binärzahl.
  Schreib-/Lesekopf auf der ersten Ziffer (von links).

Übung 3: Binär zu unär

- Eingabe: Binärzahl, little-endian 111001 (39).
  Schreib-/Lesekopf auf der ersten Ziffer (von links).
- Ausgabe: die selbe Zahl im Unärformat.
  Schreib-/Lesekopf auf der ersten 1 (von links). Die Unärzahl ist ausschliesslich von Blanks umgeben

Ideen für weitere Übungen

- Multiplikation oder Division (unär oder binär)
- Erzeugung der Folge 1.11.111.1111.11111. ...
- Erkennung der Symbolflge 'ANANAS'
- Berechnung der n-ten Fibbonacci Zahl
- Berechnung von $n!$
- Sortierung von Zahlen (oder Worten)

Die universelle TM

- TM, die alle anderen TM simulieren kann
- Die Spezifikation der anderen TM ist auf dem Band gegeben, genau wie die Eingabe für diese TM.
- Problem: Wie werden diese Daten codiert?

"Let UTM(m, n) be the class of universal Turing machine with m states and n symbols. Universal Turing machines are proved to exist in the following classes: UTM(24,2), UTM(10,3), UTM(7,4), UTM(5,5), UTM(4,6), UTM(3,10) and UTM(2,18)." *Small universal Turing machines, Yurii Rogozhin*.

Notation

- U universelle TM
- M zu simulierende TM
- T Eingabe für M

Es gibt viele Möglichkeiten, M und T zu codieren (und damit U zu programmieren).
Wir betrachten nur eine davon.

Codierung von T

Separiert durch jeweils einen Blank, um die aktuelle Position von M mit einem '!' markieren zu können.

Beispiel:
`110101` wird zu `!1.1.0.1.0.1`,
wenn sich der Schreib-/Lesekopf von M auf der ersten 1 befindet.

Kodierung einer Zustandszeile von M

Kodierung einer Zustandszeile für Symbol i:

$l_i = i\ o\ d\ n\ .$ où

- i ist das gelesene Symbol (input)
- o ist das geschriebene Symbol (output)
- d ist die Richtung (R, L, N)
- n ist die Nummer (0,. . . ,n-1) des Nachfolgezustands in Binärdarstellung (big-endian; nichts, wenn stop).

Beispiele:

`1 1 R 1 0 1 .`

read 1, write 1, go right, next state 5.

`1 0 L .`

read 1, write 0, go left, **stop**

Codierung von M

Codierung eines Zustands j:

$e_j = > \ . \ l_{i_1} \ l_{i_2} \ \ldots \ l_{i_a}$

Beispiel:

`>..0R10.00L110.11L110.>..1R10.00L111.11L111.`

Codierung von M:

$> \ e_0 \ e_1 \ \ldots \ e_n$

Implementierung

- Der aktuelle Zustand wird durch ein '!' direkt nach dem '>' markiert
- Das markierte Symbol r von T wird gesucht
- Im aktuellen Zustand wird die Zeile für r gesucht
- Entfernen der Zustandsmarkierung '!' und Markierung der Zeile durch '!'
- Speichern des zu schreibenden Symbols w und der Richtung d
- Schreiben von w auf T und Verschiebung der Markierung '!' gemäss d.
- Kopieren der Nummer des Nachfolgezustands (vor '>>').
- Entfernen von '!', um es an die richtige Stelle zu verschieben.
- Wiederherstellen des '.' vor '>>', um erneut einen Verarbeitungsschritt durchzuführen.