Einführung in die Informatik

Turing Machines

Eine abstrakte Maschine
zur Präzisierung des Algorithmenbegriffs

Wolfram Burgard
Cyrill Stachniss

Motivation und Einleitung

- Bisher haben wir **verschiedene Programmiersprachen zur Formulierung von Algorithmen** kennen gelernt.
- Dabei haben wir uns keine Gedanken gemacht über die Frage, ob diese Sprachen eigentlich äquivalent sind oder ob eine **Sprache ggf. mächtiger ist als eine andere**.
- Einen entscheidenden Beitrag zur Beantwortung dieser Frage hat **Alan Turing** geliefert.
- Mit seiner **Turing-Maschine** hat er ein **mathematisch einfaches Modell für Algorithmen** entwickelt.
- Tatsächlich hat sich gezeigt, dass **alle gängigen Programmiersprachen genau das können, was eine Turing-Maschine kann**, so dass die Menge der durch Maschinen berechenbaren Funktionen genau mit den durch Turing-Maschinen berechenbaren zusammenfallen.

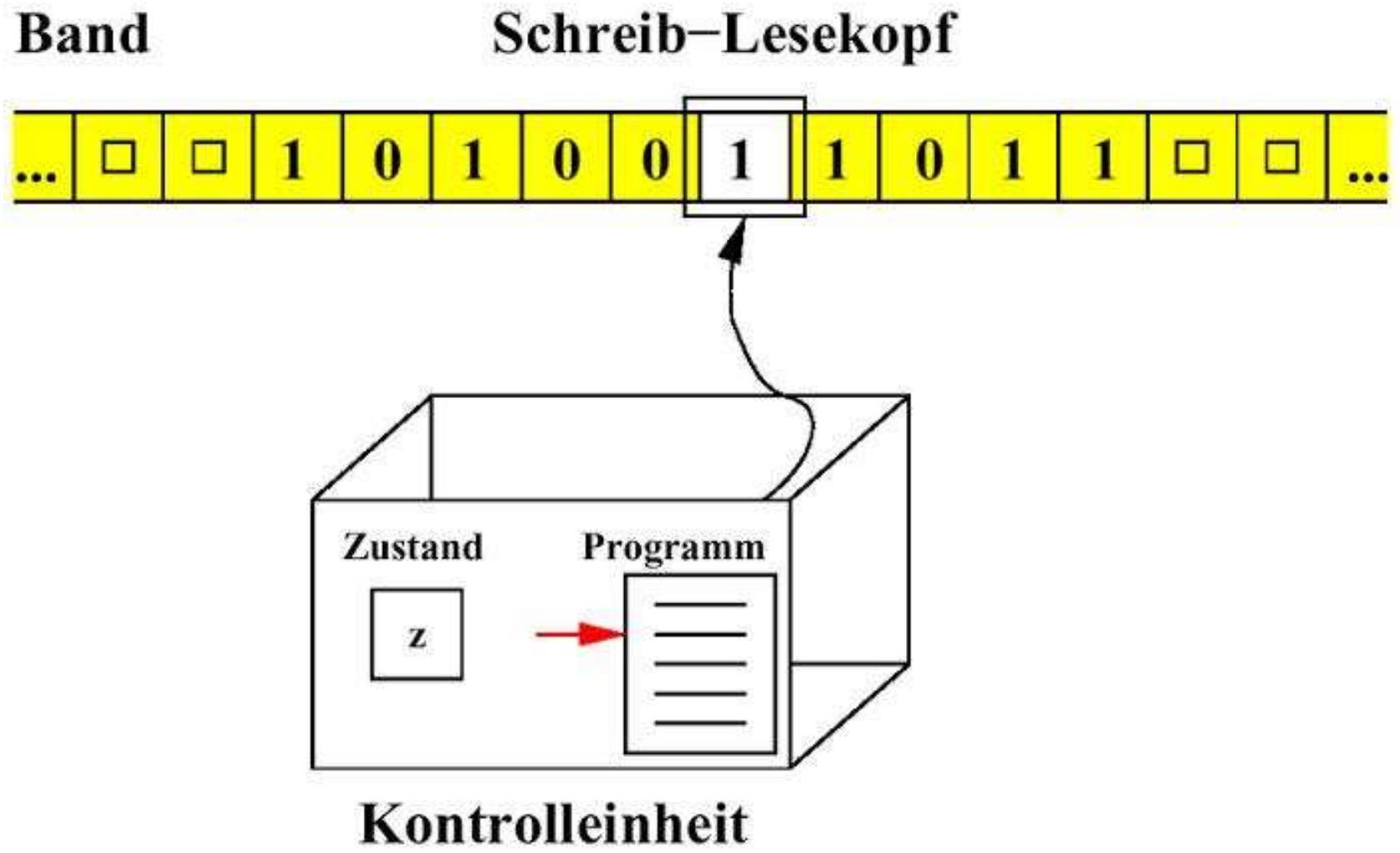
Motivation der Turing-Maschine

- Bei der Definition der Turing-Maschine ging Turing davon aus, wie Menschen eine **systematische Berechnung** (etwa eine Addition) durchführen.
- Der Mensch verwendet dafür ein **Rechenblatt**, auf dem die Rechnung einschließlich aller Rechenergebnisse notiert wird.
- Für das Aufschreiben verwendet er **Bleistift und ggf. Radiergummi**, um **Zeichen zu notieren bzw. wieder zu löschen**.
- Die **jeweilige Aktion hängt nur von endlich vielen Zeichen ab**, die sich auf dem Rechenblatt befinden.
- Die **Berechnung** selbst wird **gesteuert durch eine endliche Berechnungsvorschrift**.

Bestandteile der Turing-Maschine

- Eine **Turing-Maschine** ist eine einfache Maschine.
- Sie besteht aus:
  - einem **potentiell unendlichen Band**, welches **in Felder eingeteilt** ist und **pro Feld genau ein Zeichen** aufnehmen kann,
  - einem **Schreib-Lesekopf** sowie
  - einem **internen Zustand**.
- Je nach Zustand und Inhalt des Bandes kann die Turing-Maschine folgende Aktionen ausführen:
  - ein **neues Zeichen Schreiben**,
  - den **Schreib-Lesekopf um eine Position nach rechts oder links bewegen** und
  - den **internen Zustand verändern**.

Aufbau der Turing-Maschine



Definition der Turing-Maschine

Eine Turing-Maschine ist gegeben durch

- eine **endliche Zustandsmenge** Z,
- eine **endliche Menge von Eingabezeichen** Σ,
- eine **Menge von zulässigen Bandzeichen** $\Gamma \supset \Sigma$,
- den **Startzustand** $z_0 \in Z$,
- das **Leerzeichen** $\square \in \Gamma - \Sigma$,
- den **Endzustand** $z_e \in Z$ sowie
- die **Transitionstabelle** $\Delta \subseteq Z \times \Gamma \times \Gamma \times \{l, r, n\} \times Z$
- Insgesamt benötigen wir also ein **7-Tupel der Form**

  $(Z, \Sigma, \Gamma, z_0, \square, z_e, \Delta)$.

Bedeutung der Transitionstabelle

Die Transitionstabelle einer Turing-Maschine besteht aus 5-Tupeln der Form

$$(z, \gamma, \gamma', a, z')$$

wobei

- z der aktuelle Zustand der Maschine,
- γ das Zeichen unter dem Schreib-Lesekopf,
- a die auszuführende Aktion (`l` , `r` oder `n` für links, rechts oder noop),
- γ' das vorher zu druckende Zeichen, und
- z' der Nachfolgezustand ist.

Ein Beispielprogramm

Gegeben sei folgende Maschine:

- $Z = \{z_0, z_e\}$,
- $\sum = \{a, b\}$,
- $\Gamma = \{a, b, \square\}$,
- sowie die Transitionstabelle $\Delta =$

z_0	a	b	r	z_0
z_0	b	a	r	z_0
z_0	$\square$	$\square$	n	z_e

Wirkung dieses Programms

- Im Folgenden gehen wir immer davon aus, dass der **Schreib-Lesekopf zu Beginn stets links auf dem ersten Zeichen der Eingabe** steht und sich im **Startzustand z_0** befindet.
- Wenn dieses Programm gestartet wird, wandert die Maschine nach rechts über die Eingabe und dreht alle Zeichen um.
- D.h. für jedes `a` wird ein `b` und für jedes `b` wird ein `a` gedruckt.
- Sobald die Maschine am Ende der Eingabe angekommen ist, stoppt sie.

Ein weiteres Beispiel: Inkrementieren einer Binärzahl

Um zu einer Binärzahl eins zu addieren, gehe folgendermaßen vor:

1. Gehe zur letzten Ziffer der Zahl und starte mit einem Übertrag von `1`.
2. Steht an der aktuellen Position eine `0` und ist der Übertrag `1`, drucke eine `1`, setzen den Übertrag auf `0` und gehe nach links.
3. Steht an der aktuellen Position eine `1` und ist der Übertrag `1`, drucke eine `0`, setze den Übertrag auf `1` und gehen nach links.
4. Steht an der aktuellen Position ein Leerzeichen und ist der Übertrag `1`, drucke eine `1` und halte an.
5. Ist der Übertrag `0` und steht an der aktuellen Position eine `0` oder `1`, drucke dasselbe Zeichen und gehe nach links.
6. Ist der Übertrag `0` und steht an der aktuellen Position ein Leerzeichen, gehe nach rechts.

Am Ende steht die Maschine dann auf dem ersten Zeichen des Ergebnisses.

Wie können wir zur letzten Ziffer gehen?

Wir gehen davon aus, dass der Schreib-/Lesekopf auf dem ersten Zeichen der Eingabe steht. Allerdings funktioniert die Prozedur auch, falls die Anfangsposition ein beliebiges Zeichen ist.

1. Wenn wir eine `1` lesen, gehen drucken wir eine `1` und gehen nach rechts.

2. Wenn wir eine `0` lesen, drucken wir eine `0` und gehen nach rechts.

3. Wenn wir das Leerzeichen lesen, drucken wir das Leerzeichen und gehen nach links.

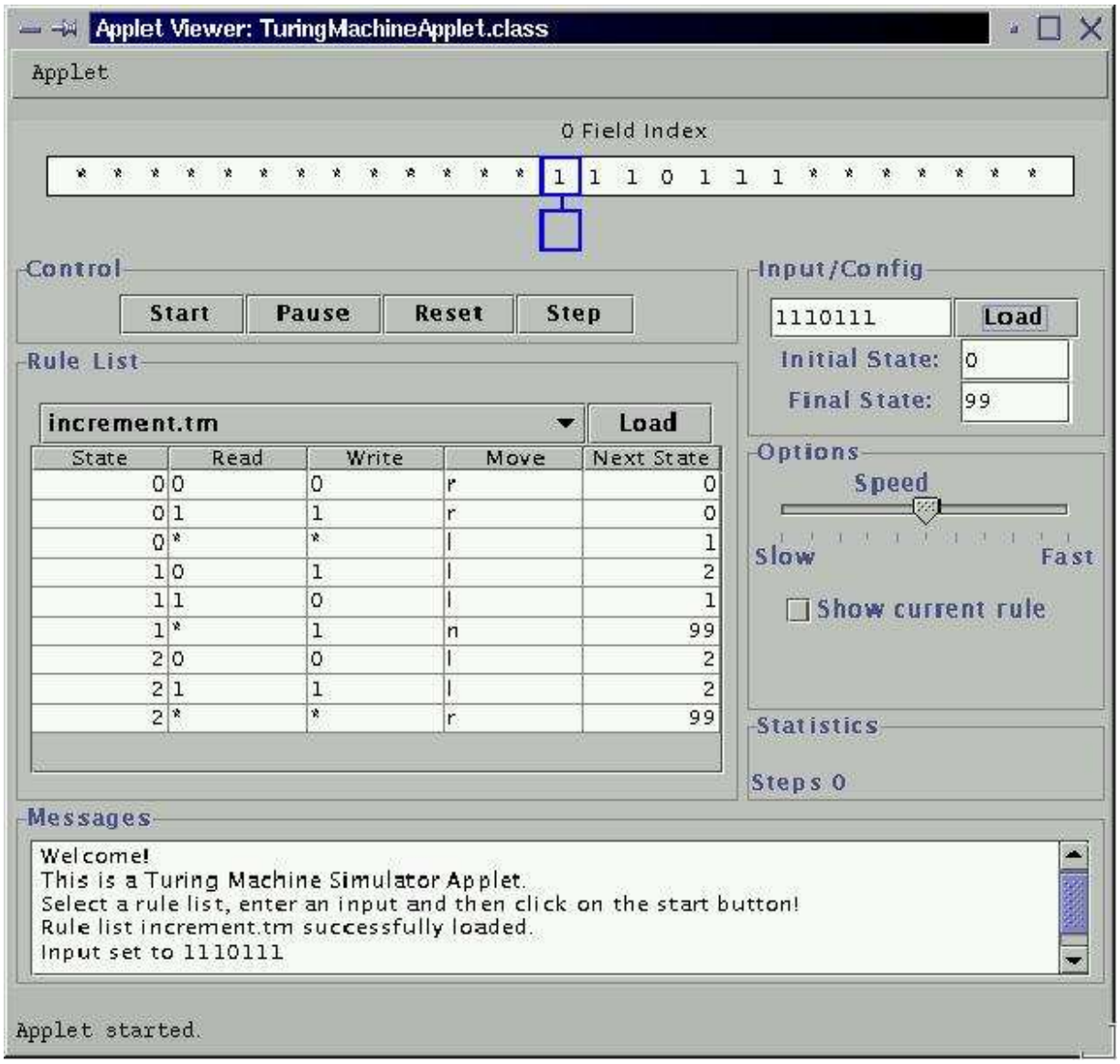
Am Ende steht die Maschine dann auf dem rechtesten Zeichen der Eingabe.

Die Transitionstabelle

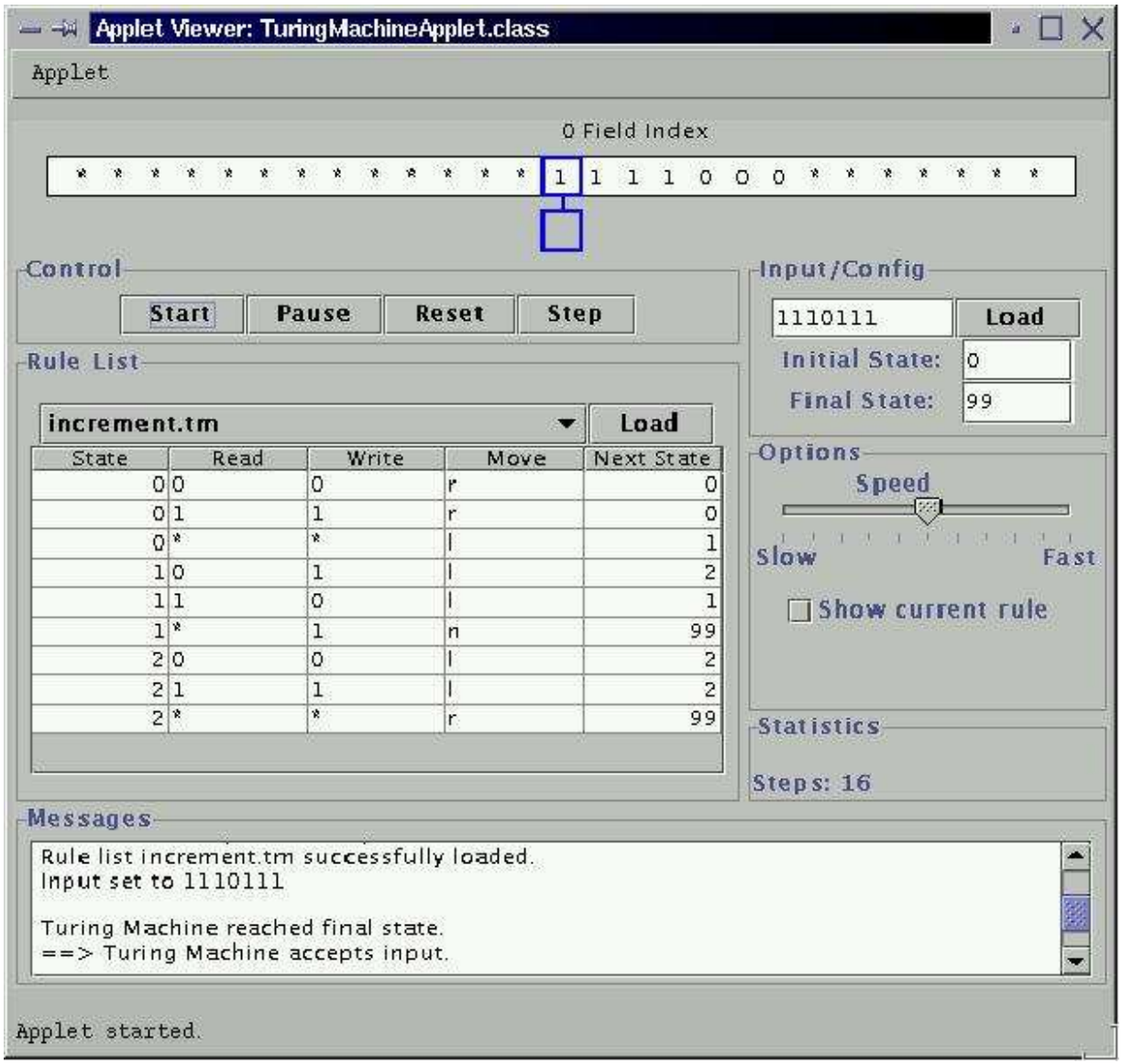
Startzustand: 0, Endzustand: 99, Leerzeichen $*$

z_i	γ_j	γ_k	A	z_l	
0	0	0	r	0	gehe nach rechts
0	1	1	r	0	
0	$*$	$*$	l	1	
1	0	1	l	2	Übertrag berechnen
1	1	0	l	1	
1	$*$	1	n	99	
2	0	0	l	2	gehe nach links
2	1	1	l	2	
2	$*$	$*$	r	99	

Anwendungsbeispiel (1)



Anwendungsbeispiel (2)

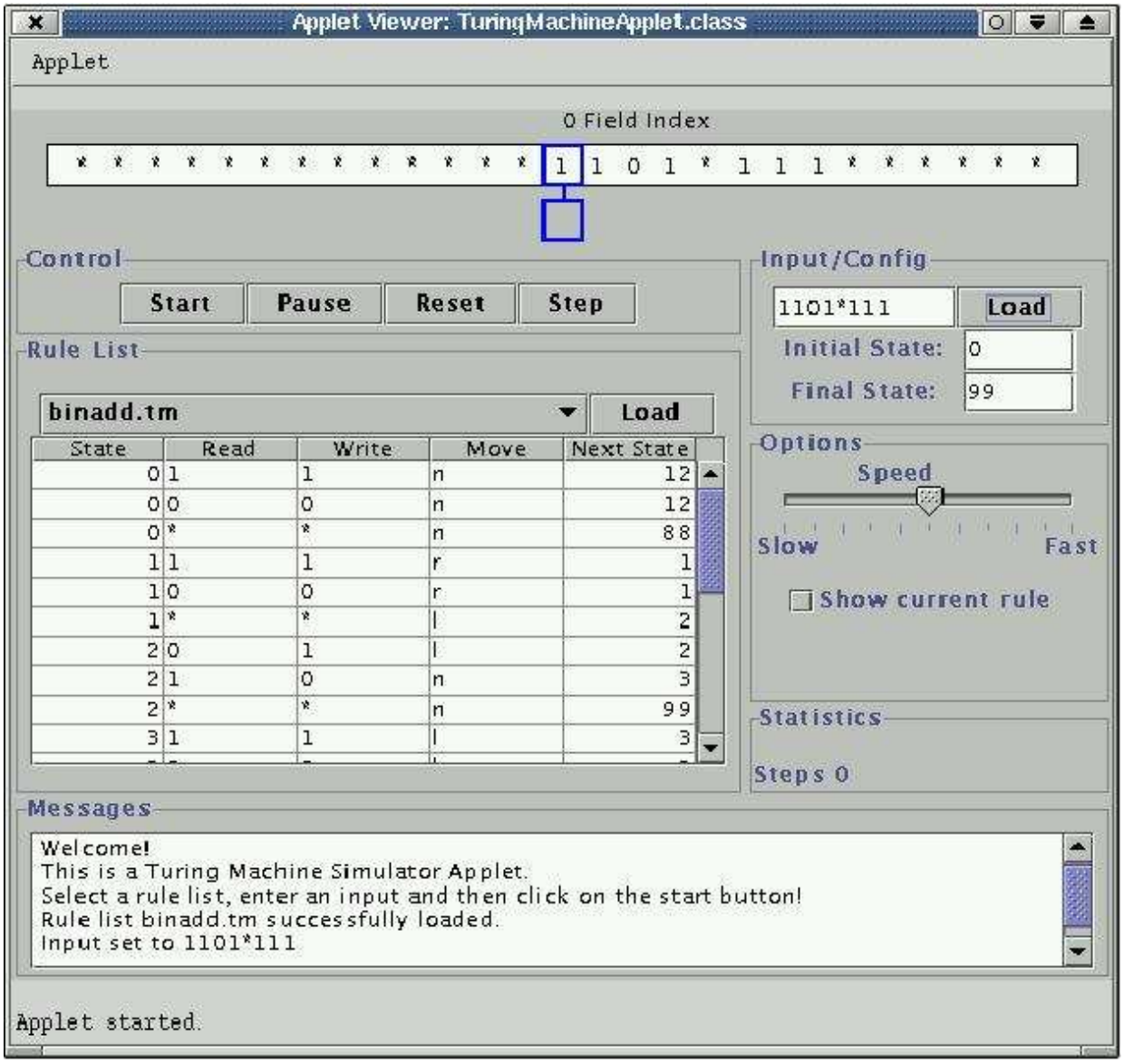


Ein komplexeres Beispiel: Addition von Binärzahlen

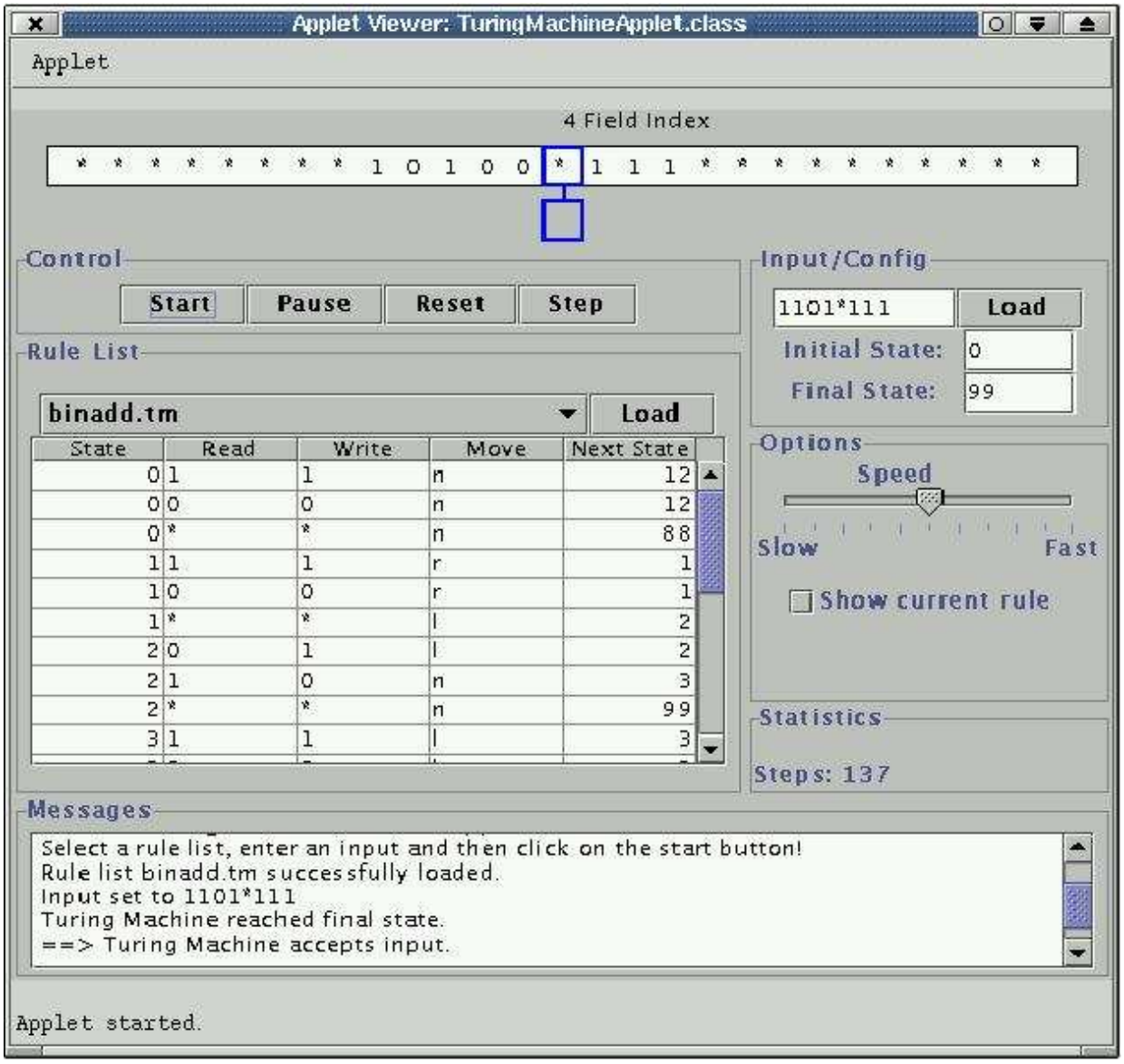
```
#          Zustaende:
#          0          Startzustand
#          88         Fehler
#          99         Endzustand
#
0          1          1          n          12
0          0          0          n          12
0          *          *          n          88
#
# Dekrementieren
#
1          1          1          r          1
1          0          0          r          1
1          *          *          l          2
#
2          0          1          l          2
2          1          0          n          3
2          *          *          n          99
#
3          1          1          l          3
3          0          0          l          3
3          *          *          n          4
#
4          *          *          l          4
4          1          1          n          11
4          0          0          n          11
```

```
#
# Inkrementieren
#
11         1          0          l          11
11         *          1          r          12
11         0          1          r          12
#
12         1          1          r          12
12         0          0          r          12
12         *          *          n          13
#
13         *          *          r          13
13         0          0          n          1
13         1          1          n          1
```

Anwendung des Addierers (Input)



Anwendungsbeispiel (Output)



Anmerkungen zum Java Applet

- Das Applet zum Simulieren von Turing-Maschinen wurde komplett in Java implementiert.
- Dabei wurden das Java2 Software-Development-Kit in der Version 1.3.1 verwendet.
- Die Graphische Oberfläche wurde mit Netbeans-IDE (Version 3.1) entwickelt.
- Nähere Infos zum Applet (Download, Installation etc.) sind zu finden unter

  `http://ais.informatik.uni-freiburg.de/turing-applet/`

Analogie zwischen manueller Berechnung und Berechnung durch die Turing-Maschine

- Offensichtlich stellt die Turing-Maschine eine Vereinfachung dar.
- Das zweidimensionale Rechenblatt wird ersetzt durch ein eindimensionales Band.
- Bleistift und Radiergummi werden durch den Schreib-Lesekopf ersetzt.
- Eine Berechnung wird nun dadurch ausgeführt, dass man den Inhalt des Bandes als Eingabe auffasst. Die Ausgabe kann man dann von dem Band ablesen, sobald die Maschine ihren Endzustand erreicht hat.

Turing-Berechenbarkeit

- Im Laufe dieser Vorlesung haben wir drei verschiedene Konzepte zur Beschreibung von Algorithmen kennengelernt.
- Dies waren Java-, while- und Turing-Maschinen-Programme.
- Alle diese Programme berechneten (ebenso wie Algorithmen) Funktionen, d.h. ermitteln für eine gegebene Eingabe eine entsprechende Ausgabe.
- Es stellt sich nun die Frage nach der Mächtigkeit dieser Beschreibungsmöglichkeiten. Oder anders ausgedrückt: Kann ich mit einem Verfahren mehr berechnen als mit einem anderen?
- Bis heute hat man noch kein Beispiel für eine berechenbare Funktion gefunden, die nicht durch eine Turing-Maschine berechenbar ist.
- Daher ist man heute davon überzeugt, dass eine Funktion, die nicht durch eine Turing-Maschine berechenbar ist, überhaupt nicht berechenbar ist.

Die Church‘sche These

Die Überzeugung, dass alles, was berechenbar ist, durch Turing-Maschinen beschreibbar ist, fasst man unter dem Namen **Church‘sche These** zusammen:

Jede im intuitiven Sinn berechenbare Funktion ist Turing-Maschinen berechenbar.

Da Turing-Maschinen Algorithmen beschreiben, können alle berechenbaren Funktionen genau durch den Begriff **Algorithmus** charakterisiert werden.

Auswirkung auf Java

- Oben haben wir gesehen, dass alles, was berechenbar ist, durch Turing-Maschinen berechnet werden kann.
- Andererseits haben wir aber ein Java-Programm gesehen, welches beliebige Turing-Maschinen-Programme ausführen kann.
- Dies war unser Applet, welches ein Turing-Maschinen-Programm einliest und auf eine beliebige Eingabe anwendet.
- Damit ist **Java mindestens so mächtig wie Turing-Maschinen**.
- Da Turing-Maschinen aber bereits das charakterisieren, was berechenbar ist, handelt es sich bei **Java** um eine **berechnungs-universelle Sprache**, d.h. wir können damit genau die berechenbaren Funktionen programmieren.
- Dies gilt übrigens auch für andere (und alle gängigen) Programmiersprachen, wie z.B. Pascal, Lisp, Scheme, Prolog, Simula, Fortran, C, C++, Perl, Python, ...

Nicht-Determinismus und Turing-Maschinen

- An Turing-Maschinen lässt sich der Begriff des **Nicht-Determinismus** sehr gut deutlich machen.
- Wir haben nämlich **nicht festgelegt**, dass es für **jede Kombination von aktuellem Zustand und Bandsymbol** nur **genau einen Eintrag in der Programmtabelle** geben darf.
- Tatsächlich sind **mehrere Aktionen** für einen Zustand und ein Bandsymbol **möglich**.
- In diesem Fall wird die Turing-Maschine **nicht-deterministisch**, denn sie kann einen beliebigen, passenden Eintrag aus der Tabelle verwenden.
- Eine **nicht-deterministische Turing-Maschine** kann somit eine **Lösung zufällig schnell ermitteln**.
- Allerdings kann man **mit nicht-deterministischen Turing-Maschinen nicht mehr Probleme lösen als mit deterministischen** .

Zusammenfassung

- **Turing-Maschinen** stellen eine einfache, abstrakte Art von Maschinen dar.
- Sie wurden entwickelt, um den **Begriff des algorithmisch berechenbaren zu charakterisieren**.
- Trotz ihrer einfachen Konstruktion sind Turing-Maschinen **berechnungs-universell**, d.h. sie können jede im intuitiven Sinn berechenbare Funktion berechnen.
- Dies wir durch die **Church‘sche These** untermauert.
- Da man **mit Java beliebige Turing-Maschinen realisieren und simulieren** kann, ist auch **Java berechnungsuniversell**.
- Damit können wir prinzipiell **jeden Algorithmus in Java realisieren**.