Informatik 1

Wintersemester 2007/2008

Helmut Seidl

Institut für Informatik
TU München

0 Allgemeines

Inhalt dieser Vorlesung:

- Einführung in Grundkonzepte der Informatik;
- Einführung in Denkweisen der Informatik;
- Programmieren in Java :-)

Voraussetzungen:

Informatik Leistungkurs: nützlich, aber nicht nötig ;-)

Kenntnis einer Programmiersprache:
nützlich, aber nicht nötig :-)

Eigener Rechner: nützlich, aber nicht nötig :-)

Voraussetzungen:

Informatik Leistungkurs:	nützlich, aber nicht nötig ;-)
Kenntnis einer Programmiersprache:	nützlich, aber nicht nötig :-)
Eigener Rechner:	nützlich, aber nicht nötig :-)
Abstraktes Denken:	unbedingt erforderlich !!!
Neugierde, technisches Interesse:	unbedingt erforderlich !!!

Unsere Philosophie:

Schreiben ist Macht

Eine alte Kulturtechnik:

- um Wissen haltbar zu machen;
- neue Denktechniken zu entwickeln ...

Schreiben als politisches Instrument:

$\Rightarrow$ um zu bestimmen, was Realität ist;

$\Rightarrow$ um das Handeln von Menschen zu beeinflussen ...

(Vorschriften, Gesetze)

Schreiben als praktisches Werkzeug:

$\Rightarrow$ um festzulegen, wie ein Vorgang ablaufen soll:

- Wie soll das Regal zusammen gebaut werden?
- Wie multipliziert man zwei Zahlen?

$\Rightarrow$ um das Verhalten komplexer mit ihrer Umwelt interagierender Systeme zu realisieren

(etwa Motorsteuerungen, Roboter)

$\Longrightarrow$ Programmierung

1 Vom Problem zum Programm

Ein Problem besteht darin, aus einer gegebenen Menge von Informationen eine weitere (bisher unbekannte) Information zu bestimmen.

Ein Algorithmus ist ein exaktes Verfahren zur Lösung eines Problems, d.h. zur Bestimmung der gewünschten Resultate.

$\Longrightarrow$

Ein Algorithmus beschreibt eine Funktion: $f : E \to A$,
wobei E = zulässige Eingaben, A = mögliche Ausgaben.

Abu Abdallah Muhamed ibn Musa al'Khwaritzmi, etwa 780–835

Achtung:

Nicht jede Abbildung lässt sich durch einen Algorithmus realisieren! (↑Berechenbarkeitstheorie)

Das Verfahren besteht i.a. darin, eine Abfolge von Einzelschritten der Verarbeitung festzulegen.

Beispiel: Alltagsalgorithmen

Resultat	Algorithmus	Einzelschritte
Pullover	Strickmuster	eine links, eine rechts eine fallen lassen
Kuchen	Rezept	nimm 3 Eier ...
Konzert	Partitur	Noten

Beispiel: Euklidischer Algorithmus

Problem: Seien $a, b \in \mathbb{N}, a, b \neq 0$. Bestimme $ggT(a, b)$.

Beispiel: Euklidischer Algorithmus

Problem: Seien $a, b \in \mathbb{N}, a, b \neq 0$. Bestimme $ggT(a, b)$.

Algorithmus:

1. Falls $a = b$, brich Berechnung ab, es gilt $ggT(a, b) = a$.
   Ansonsten gehe zu Schritt 2.
2. Falls $a > b$, ersetze a durch $a - b$ und
   setze Berechnung in Schritt 1 fort.
   Ansonsten gehe zu Schritt 3.
3. Es gilt $a < b$. Ersetze b durch $b - a$ und
   setze Berechnung in Schritt 1 fort.

Eigenschaften von Algorithmen:

Abstrahierung: Allgemein löst ein Algorithmus eine Klasse von Problem-Instanzen. Die Anwendung auf eine konkrete Aufgabe erfordert Abstraktion :-)

Determiniertheit: Algorithmen sind im allgemeinen determiniert, d.h. mit gleichen Eingabedaten und gleichem Startzustand wird stets ein gleiches Ergebnis geliefert. (↑ nichtdeterministische Algorithmen, ↑randomisierte Algorithmen)

Finitheit: Die Beschreibung eines Algorithmus besitzt endliche Länge. Die bei der Abarbeitung eines Algorithmus entstehenden Datenstrukturen und Zwischenergebnisse sind endlich.

Terminierung: Algorithmen, die nach endlich vielen Schritten ein Resultat liefern, heißen terminierend. Meist sind nur terminierende Algorithmen von Interesse. Ausnahmen: Betriebssysteme, reaktive Systeme, ...

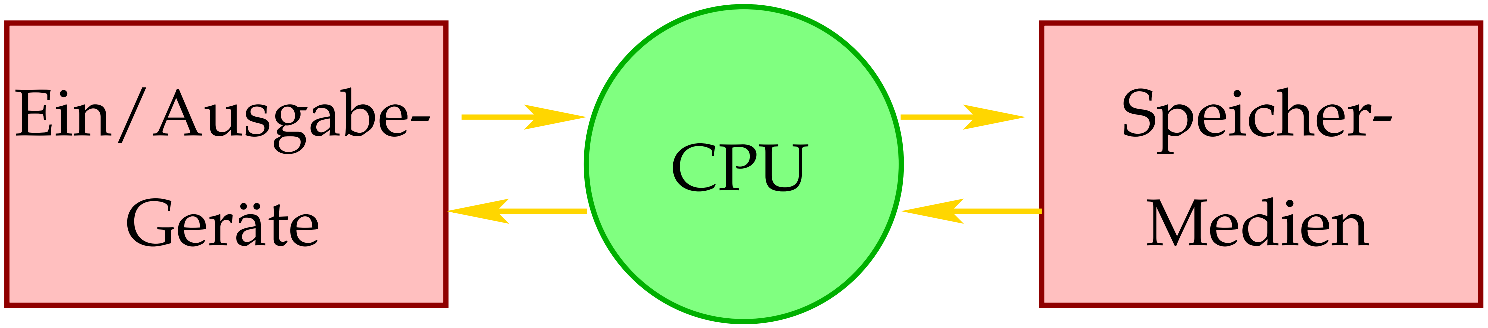
Ein Programm ist die Formulierung eines Algorithmus in einer Programmiersprache.

Die Formulierung gestattet (hoffentlich :-) eine maschinelle Ausgeführung.

Beachte:

- Es gibt viele Programmiersprachen: Java, C, Prolog, Fortran, Cobol ....
- Eine Programmiersprache ist dann gut, wenn
  - die Programmiererin in ihr ihre algorithmischen Ideen natürlich beschreiben kann, insbesondere selbst später noch versteht, was das Programm tut (oder nicht tut);
  - ein Computer das Programm leicht verstehen und effizient ausführen kann.

Typischer Aufbau eines Computers:



Ein/Ausgabegeräte (= input/output devices) — ermöglichen Eingabe des Programms und der Daten, Ausgabe der Resultate.

CPU (= central processing unit) — führt Programme aus.

Speicher-Medien (= memory) — enthalten das Programm sowie die während der Ausführung benötigten Daten.

Hardware $==$ physikalische Bestandteile eines Computers.

Merkmale von Computern:

Geschwindigkeit: schnelle Ausgeführung auch komplexer Programme.

Zuverlässigkeit: Hardwarefehler sind selten :-)
Fehlerhafte Programme bzw. falsche Eingaben sind häufig :-(

Speicherkapazität: riesige Datenmengen speicherbar und schnell zugreifbar.

Kosten: Niedrige laufende Kosten.

Algorithmen wie Programme abstrahieren von (nicht so wesentlichen) Merkmalen realer Hardware.

$\Longrightarrow$ Annahme eines (nicht ganz realistischen, dafür exakt definierten) Maschinenmodells für die Programmierung.

Beliebte Maschinenmodelle:

Turingmaschine: eine Art Lochstreifen-Maschine
(Turing, 1936 :-)

Registermaschine: etwas realistischerer Rechner, allerdings mit i.a. beliebig großen Zahlen und unendlich viel Speicher;

λ-Kalkül: eine minimale ↑funktionale Programmiersprache;

JVM: (Java-Virtual Machine) – die virtuelle Maschine für Java (↑Compilerbau);

...

Zur Definition eines Maschinenmodells benötigen wir:

- Angabe der zulässigen Datenobjekte/Speicherbereiche, auf denen Operationen ausgeführt werden sollen;
- Angabe der verfügbaren Einzelschritte / Aktionen / Elementaroperationen;
- Angabe der Kontrollstrukturen zur Angabe der beabsichtigten Ausführungsreihenfolgen.

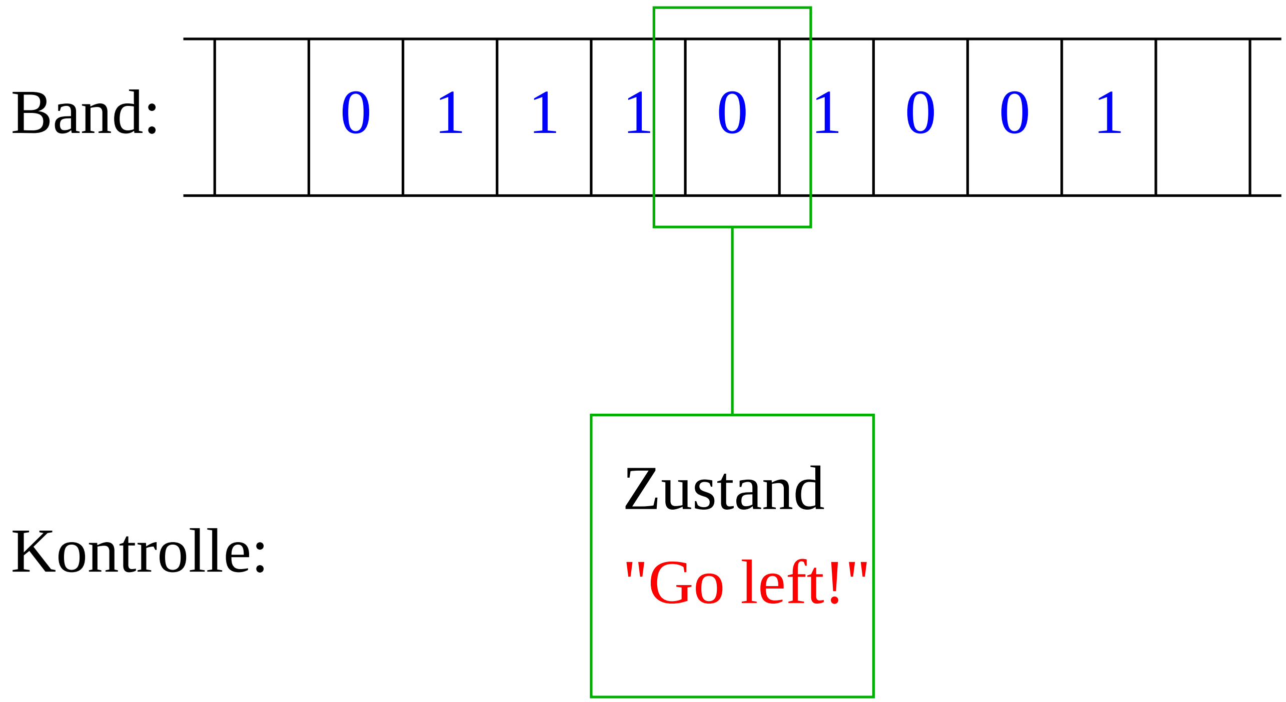
Beispiel 1: Turing-Maschine

Alan Turing, 1912–1954

Daten der Turing-Maschine: Eine Folge von 0 und 1 und evt. weiterer Symbole wie z.B. “ ” (Blank – Leerzeichen) auf einem Band zusammen mit einer Position des “Schreib/Lese”-Kopfs auf dem Band;

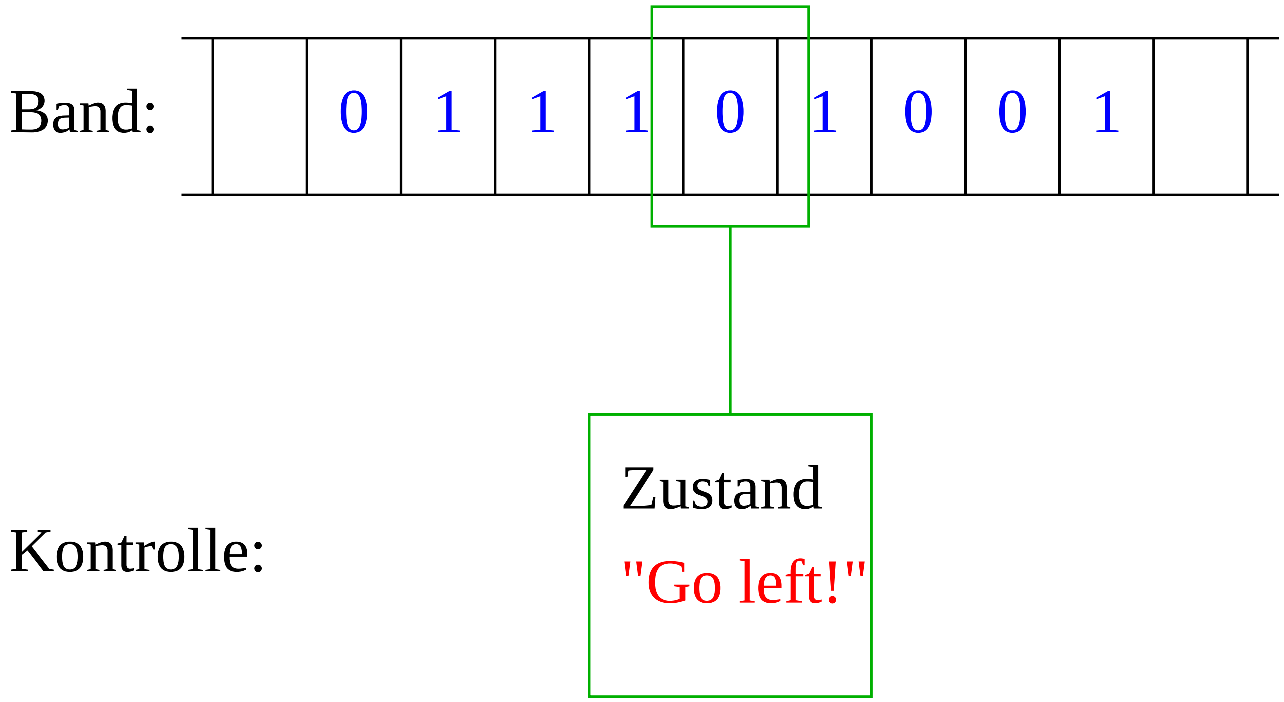
Operationen: Überschreiben des aktuellen Zeichens und Verrücken des Kopfs um eine Position nach rechts oder links;

Kontrollstrukturen: Es gibt eine endliche Menge Q von Zuständen. In Abhängigkeit vom aktuellen Zustand und dem gelesenen Zeichen wird die Operation ausgewählt – und der Zustand geändert.



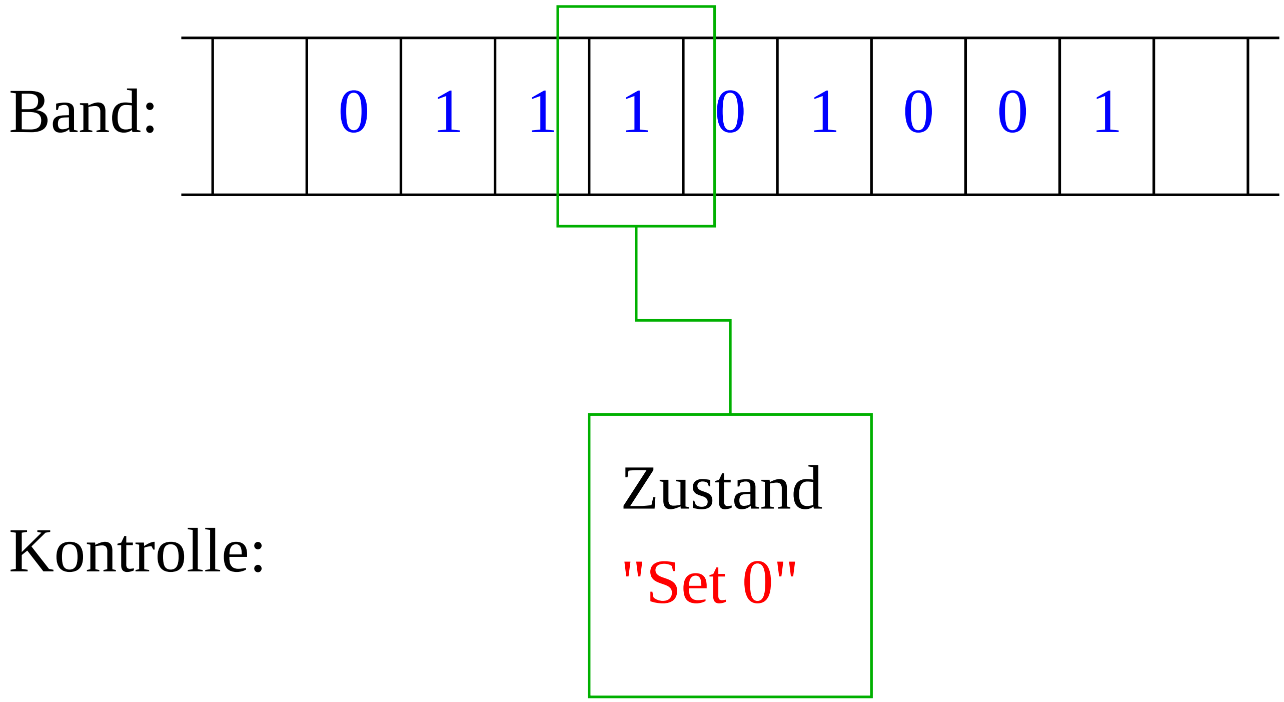
Programm:

Zustand	Input	Operation		neuer Zustand
"Go left!"	0	0	links	"Set 0"
"Go left!"	1	1	rechts	"Go left!"
"Set 0"	0	0	–	"Stop"
"Set 0"	1	0	links	"Set 0"

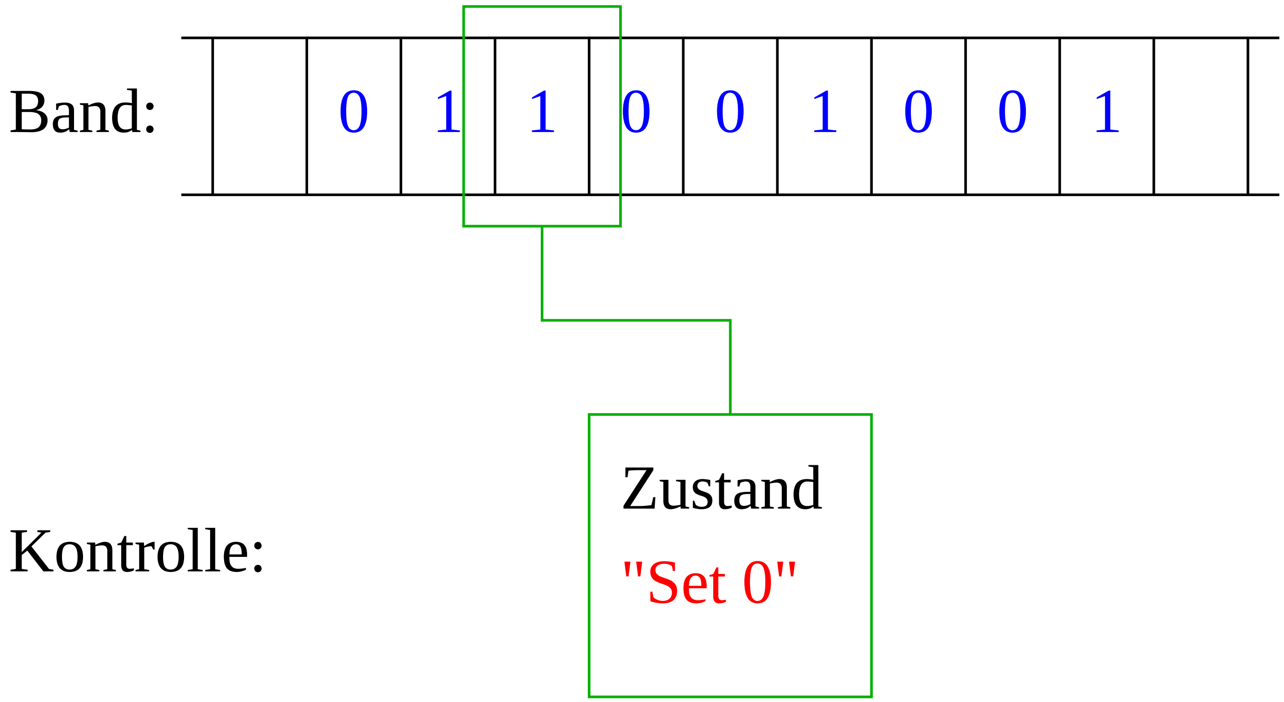


Operation = "Schreibe eine 0 und gehe nach links!"

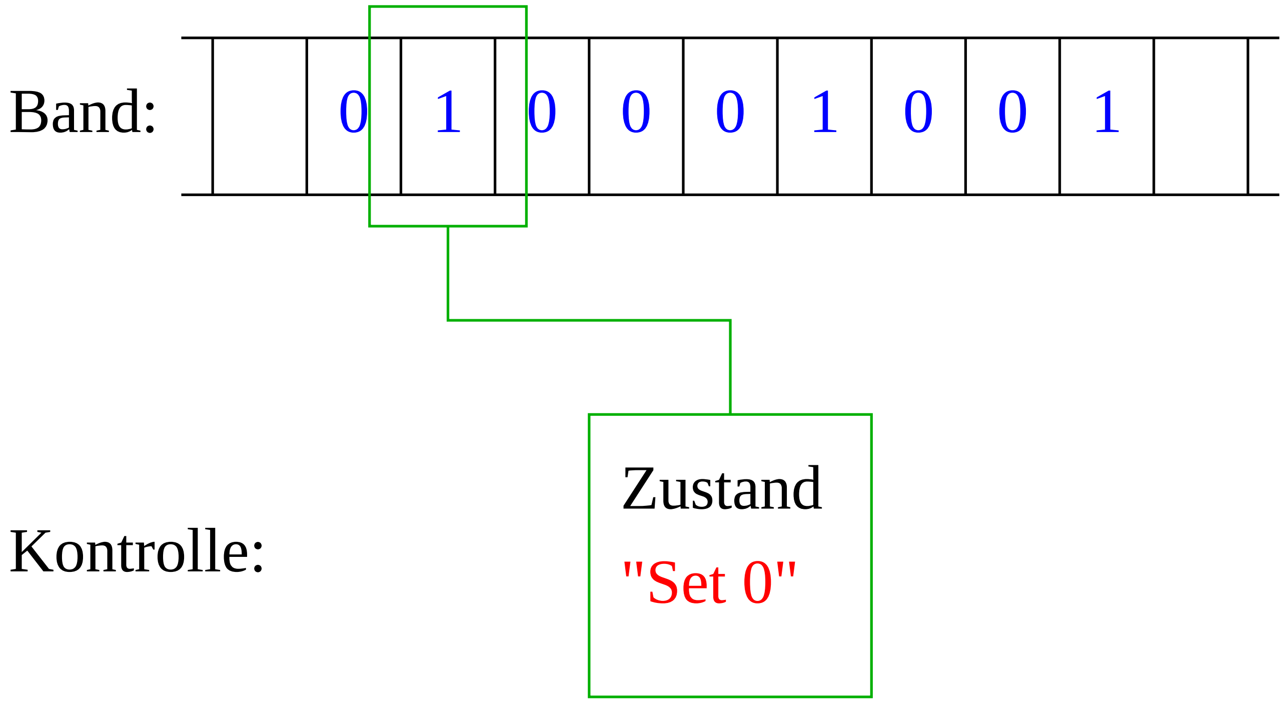
neuer Zustand = "Set 0"



Operation	= “Schreibe eine 0 und gehe nach links!”
neuer Zustand	= unverändert

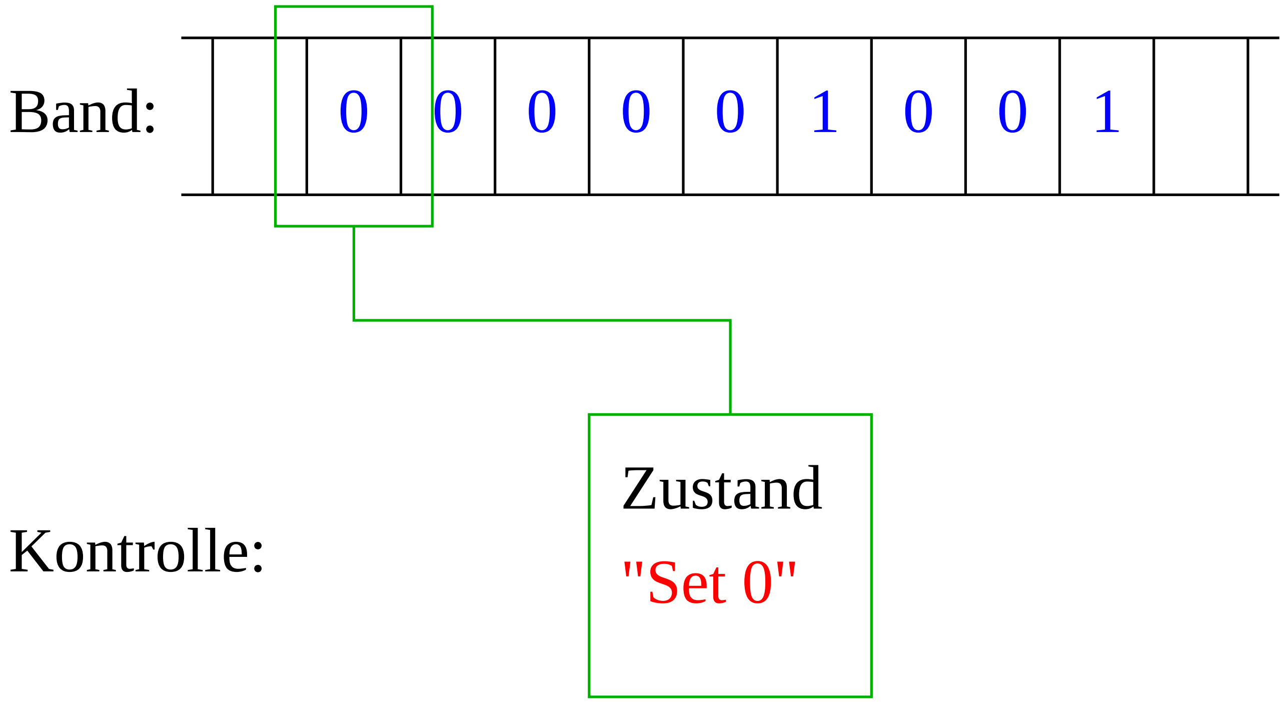


Operation	= “Schreibe eine 0 und gehe nach links!”
neuer Zustand	= unverändert



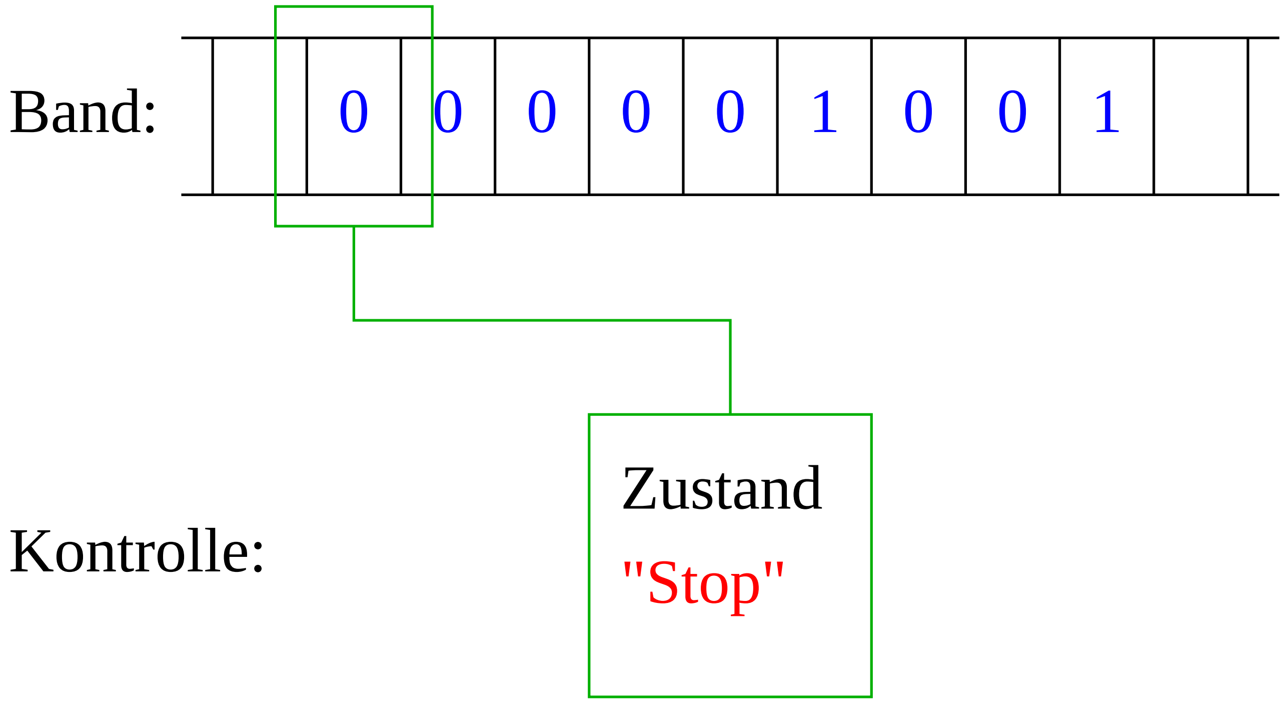
Operation = "Schreibe eine 0 und gehe nach links!"

neuer Zustand = unverändert



Operation = keine

neuer Zustand = “Stop”



Ende der Berechnung.

Fazit:

Die Turing-Maschine ist

- ... sehr einfach;
- ... sehr mühsam zu programmieren;
- ... aber nichtsdestoweniger universell, d.h. prinzipiell in der Lage alles zu berechnen, d.h. insbesondere alles, was ein Desktop-Rechner kann :-)

$\Longrightarrow$ beliebtes Hilfsmittel in der ↑Berechenbarkeitstheorie und in der ↑Komplexitätstheorie.

Beispiel 2: JVM

- minimale Menge von Operationen, Kontroll- sowie Datenstrukturen, um Java-Programme auszuführen.

$\Longrightarrow$ Um Java auf einem Rechner XYZ auszuführen, benötigt man nur einen Simulator für die JVM, der auf XYZ läuft.

$\Longrightarrow$ Portabilität!

Ähnliche abstrakte Maschinen gibt es auch für viele andere Programmiersprachen, z.B. Pascal, SmallTalk, Prolog, SML,...
$\uparrow$Compilerbau