Script generated by TTT

Title: Seidl: Theoretische_Informatik (13.06.2013)

Date: Thu Jun 13 16:02:58 CEST 2013

Duration: 89:27 min

Pages: 66

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?
- Welche Eigenschaften von Programmen sind entscheidbar?

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?
- Welche Eigenschaften von Programmen sind entscheidbar?
  ZB Termination? Code-Erreichbarkeit? Überlauf?

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?
- Welche Eigenschaften von Programmen sind entscheidbar?
  ZB Termination? Code-Erreichbarkeit? Überlauf?
- Was kann man in polynomieller Zeit berechnen?

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?
- Welche Eigenschaften von Programmen sind entscheidbar?
  ZB Termination? Code-Erreichbarkeit? Überlauf?
- Was kann man in polynomieller Zeit berechnen?
  Mit welchen Maschinen?

4. Berechenbarkeit und Entscheidbarkeit

4.1 Der Begriff der Berechenbarkeit

4. Berechenbarkeit und Entscheidbarkeit

4.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält

4. Berechenbarkeit und Entscheidbarkeit

4.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält, falls $f(n_1, \ldots, n_k)$ definiert ist,

4. Berechenbarkeit und Entscheidbarkeit

4.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält, falls $f(n_1, \ldots, n_k)$ definiert ist,
- und nicht terminiert, falls $f(n_1, \ldots, n_k)$ nicht definiert ist.

4. Berechenbarkeit und Entscheidbarkeit

4.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält, falls $f(n_1, \ldots, n_k)$ definiert ist,
- und nicht terminiert, falls $f(n_1, \ldots, n_k)$ nicht definiert ist.

Was bedeutet „Algorithmus“? Assembler? C? Java? OCaml?

4. Berechenbarkeit und Entscheidbarkeit

4.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält, falls $f(n_1, \ldots, n_k)$ definiert ist,
- und nicht terminiert, falls $f(n_1, \ldots, n_k)$ nicht definiert ist.

Was bedeutet „Algorithmus“? Assembler? C? Java? OCaml?
Macht es einen Unterschied?

Achtung: Berechenbarkeit setzt zwei verschiedene Dinge in Beziehung:

- Algorithmen, dh endliche Wörter.
- Mathematische Funktionen, dh Mengen von Paaren.

4. Berechenbarkeit und Entscheidbarkeit

4.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält, falls $f(n_1, \ldots, n_k)$ definiert ist,
- und nicht terminiert, falls $f(n_1, \ldots, n_k)$ nicht definiert ist.

Was bedeutet „Algorithmus“? Assembler? C? Java? OCaml? Macht es einen Unterschied?

Achtung: Berechenbarkeit setzt zwei verschiedene Dinge in Beziehung:

- Algorithmen, dh endliche Wörter.
- Mathematische Funktionen, dh Mengen von Paaren.

ZB beschreibt $x \mapsto x^2$ die Funktion

$$\{(0,0), (1,1), (2,4), (3,9), (4,16), (5,25), \ldots\}$$

Terminologie: Eine Funktion $f : A \to B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

Terminologie: Eine Funktion $f : A \to B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

partiell gdw $f(a)$ auch undefiniert sein kann.

Terminologie: Eine Funktion $f : A \to B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

partiell gdw $f(a)$ auch undefiniert sein kann.

echt partiell gdw sie nicht total ist.

Terminologie: Eine Funktion $f : A \to B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

partiell gdw $f(a)$ auch undefiniert sein kann.

echt partiell gdw sie nicht total ist.

Beispiel 4.1

Der Algorithmus

```
input(n);
while true do ;
```

Terminologie: Eine Funktion $f : A \to B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

partiell gdw $f(a)$ auch undefiniert sein kann.

echt partiell gdw sie nicht total ist.

Beispiel 4.1

Der Algorithmus

```
input(n);
while true do ;
```

berechnet die überall undefinierte Funktion, dh $\emptyset \subset \mathbb{N} \to \mathbb{N}$.

Beispiel 4.2

Ist die Funktion

$$f_1(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück der Dezimalbruchentwicklung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_1(31415) = 1$ aber $f_1(315) = 0$)

Beispiel 4.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$)

Beispiel 4.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$) Unbekannt!

Beispiel 4.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$) Unbekannt!

Durch schrittweise Approximation und Suche in der Dezimalbruchentwicklung von π kann man feststellen, *dass* n vorkommt.

Beispiel 4.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$) Unbekannt!

Durch schrittweise Approximation und Suche in der Dezimalbruchentwicklung von π kann man feststellen, *dass* n vorkommt.

Aber wie stellt man fest, dass n *nicht* vorkommt?
Nichttermination statt 0!

Vielleicht gibt es aber einen (noch zu findenden) mathematischen Satz, der genaue Aussagen über die in π vorkommenden Ziffernfolgen macht.

Beispiel 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Beispiel 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor,

Beispiel 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor,
  dann ist $f_3(n) = 1$ für alle n,
- oder es gibt eine längste vorkommende Sequenz 0^m,

Beispiel 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor,
  dann ist $f_3(n) = 1$ für alle n,
- oder es gibt eine längste vorkommende Sequenz 0^m,
  dann ist $f_3(n) = 1$ für $n \leq m$ und $f_3(n) = 0$ sonst.

Beispiel 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor,
  dann ist $f_3(n) = 1$ für alle n,
- oder es gibt eine längste vorkommende Sequenz 0^m,
  dann ist $f_3(n) = 1$ für $n \leq m$ und $f_3(n) = 0$ sonst.

Beide Funktionen sind berechenbar.

Satz 4.5
Es gibt nicht-berechenbare Funktionen in $\mathbb{N} \to \{0,1\}$.

Satz 4.5
Es gibt nicht-berechenbare Funktionen in $\mathbb{N} \to \{0,1\}$.

Beweis:
Es gibt nur abzählbar viele Algorithmen, aber überabzählbar viele Funktionen in $\mathbb{N} \to \{0,1\}$ (Beweis wie zu Satz 1.8).

□

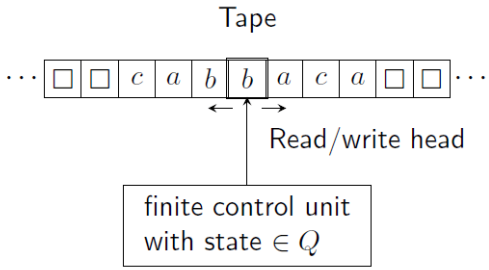
Verschiedene Formalisierungen des Begriffs der Berechenbarkeit:

- Turing-Maschinen (Turing 1936)
- λ-Kalkül (Church 1936)
- μ-rekursive Funktionen
- Markov-Algorithmen
- Registermaschinen
- Awk, Basic, C, Dylan, Eiffel, Fortran, Java, Lisp, Modula, Oberon, Pascal, Python, Ruby, Simula, TEX, ...-Programme
- while-Programme
- goto-Programme
- DNA-Algorithmen
- Quantenalgorithmen
- ...

Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

Churchsche These / Church-Turing These
Der formale Begriff der Berechenbarkeit mit Turing-Maschinen (bzw λ-Kalkül etc) stimmt mit dem intuitiven Berechenbarkeitsbegriff überein.



Definition 4.6
Eine Turingmaschine (TM) ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.

Definition 4.6
Eine Turingmaschine (TM) ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.
- Σ ist eine endliche Menge, das Eingabealphabet.
- Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ ist die Übergangsfunktion.

$a, b \in \Gamma$
$\delta(q, a) = (p, b, t)$

Definition 4.6

Eine Turingmaschine (TM) ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.
- Σ ist eine endliche Menge, das Eingabealphabet.
- Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ ist die Übergangsfunktion.
- δ darf partielle sein.

Definition 4.6

Eine Turingmaschine (TM) ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.
- Σ ist eine endliche Menge, das Eingabealphabet.
- Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ ist die Übergangsfunktion.
- δ darf partielle sein.
- $q_0 \in Q$ ist der Startzustand.
- $\square \in \Gamma \setminus \Sigma$ ist das Leerzeichen.
- $F \subseteq Q$ ist die Menge der akzeptierenden oder Endzustände.

Definition 4.6

Eine Turingmaschine (TM) ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.
- Σ ist eine endliche Menge, das Eingabealphabet.
- Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ ist die Übergangsfunktion.
- δ darf partielle sein.
- $q_0 \in Q$ ist der Startzustand.
- $\square \in \Gamma \setminus \Sigma$ ist das Leerzeichen.
- $F \subseteq Q$ ist die Menge der akzeptierenden oder Endzustände.

Eine nichtdeterministische Turingmaschine hat eine Übergangsfunktion $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$.

Intuitiv bedeutet $\delta(q, a) = (q', b, d)$:

Intuitiv bedeutet $\delta(q, a) = (q', b, d)$:

- Wenn sich M im Zustand q befindet,
- und auf dem Band a liest,

Intuitiv bedeutet $\delta(q, a) = (q', b, d)$:

- Wenn sich M im Zustand q befindet,
- und auf dem Band a liest,
- so geht M im nächsten Schritt in den Zustand q' über,
- überschreibt a mit b,

Intuitiv bedeutet $\delta(q, a) = (q', b, d)$:

- Wenn sich M im Zustand q befindet,
- und auf dem Band a liest,
- so geht M im nächsten Schritt in den Zustand q' über,
- überschreibt a mit b,
- und bewegt danach den Schreib-/Lesekopf nach rechts (falls $d = R$), nach links (falls $d = L$), oder nicht (falls $d = N$).

Definition 4.7

Eine Konfiguration einer Turingmaschine ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.

Definition 4.7
Eine Konfiguration einer Turingmaschine ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.

Dies modelliert

- Bandinhalt: $\ldots \Box\alpha\beta\Box \ldots$
- Zustand: q
- Kopf auf dem ersten Zeichen von $\beta\Box$

Definition 4.7
Eine Konfiguration einer Turingmaschine ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.

Dies modelliert

- Bandinhalt: $\ldots \Box\alpha\beta\Box \ldots$
- Zustand: q
- Kopf auf dem ersten Zeichen von $\beta\Box$

Die Startkonfiguration der Turingmaschine bei Eingabe $w \in \Sigma^*$ ist (ϵ, q_0, w).

Die Berechnung der TM M wird als Relation $\to_M$ auf Konfigurationen formalisiert. Falls $\delta(q, \mathit{first}(\beta)) = (q', c, d)$:

$$(\alpha, q, \beta) \to_M$$

Die Berechnung der TM M wird als Relation $\to_M$ auf Konfigurationen formalisiert. Falls $\delta(q, \mathit{first}(\beta)) = (q', c, d)$:

$$(\alpha, q, \beta) \to_M \begin{cases} (\alpha, q', c\ \mathit{rest}(\beta)) & \text{falls } d = N \\ \\ \\ \end{cases}$$

Die Berechnung der TM M wird als Relation $\to_M$ auf Konfigurationen formalisiert. Falls $\delta(q, \mathit{first}(\beta)) = (q', c, d)$:

$$(\alpha, q, \beta) \to_M \begin{cases} (\alpha, q', c\ \mathit{rest}(\beta)) & \text{falls } d = N \\ (\alpha c, q', \mathit{rest}(\beta)) & \text{falls } d = R \end{cases}$$

Die Berechnung der TM M wird als Relation $\to_M$ auf Konfigurationen formalisiert. Falls $\delta(q, \mathit{first}(\beta)) = (q', c, d)$:

$$(\alpha, q, \beta) \to_M \begin{cases} (\alpha, q', c\ \mathit{rest}(\beta)) & \text{falls } d = N \\ (\alpha c, q', \mathit{rest}(\beta)) & \text{falls } d = R \\ (\mathit{butlast}(\alpha), q', \mathit{last}(\alpha)\ c\ \mathit{rest}(\beta)) & \text{falls } d = L \end{cases}$$

Die Berechnung der TM M wird als Relation $\to_M$ auf Konfigurationen formalisiert. Falls $\delta(q, \mathit{first}(\beta)) = (q', c, d)$:

$$(\alpha, q, \beta) \to_M \begin{cases} (\alpha, q', c\ \mathit{rest}(\beta)) & \text{falls } d = N \\ (\alpha c, q', \mathit{rest}(\beta)) & \text{falls } d = R \\ (\mathit{butlast}(\alpha), q', \mathit{last}(\alpha)\ c\ \mathit{rest}(\beta)) & \text{falls } d = L \end{cases}$$

wobei

$$\begin{array}{ll} \mathit{first}(aw) = a & \mathit{first}(\epsilon) = \square \\ \mathit{rest}(aw) = w & \mathit{rest}(\epsilon) = \epsilon \\ \mathit{last}(wa) = a & \mathit{last}(\epsilon) = \square \\ \mathit{butlast}(wa) = w & \mathit{butlast}(\epsilon) = \epsilon \end{array}$$

für $a \in \Gamma$ und $w \in \Gamma^*$.

Die Berechnung der TM M wird als Relation $\to_M$ auf Konfigurationen formalisiert. Falls $\delta(q, \mathit{first}(\beta)) = (q', c, d)$:

$$(\alpha, q, \beta) \to_M \begin{cases} (\alpha, q', c\ \mathit{rest}(\beta)) & \text{falls } d = N \\ (\alpha c, q', \mathit{rest}(\beta)) & \text{falls } d = R \\ (\mathit{butlast}(\alpha), q', \mathit{last}(\alpha)\ c\ \mathit{rest}(\beta)) & \text{falls } d = L \end{cases}$$

wobei

$$\begin{array}{ll} \mathit{first}(aw) = a & \mathit{first}(\epsilon) = \square \\ \mathit{rest}(aw) = w & \mathit{rest}(\epsilon) = \epsilon \\ \mathit{last}(wa) = a & \mathit{last}(\epsilon) = \square \\ \mathit{butlast}(wa) = w & \mathit{butlast}(\epsilon) = \epsilon \end{array}$$

für $a \in \Gamma$ und $w \in \Gamma^*$.

Falls M nichtdeterministisch ist: $\delta(q, \mathit{first}(\beta)) \ni (q', c, d)$

Beispiel 4.8 (Unär +1)

$$M = (\{q, f\}, \{1\}, \{1, \Box\}, \delta, q, \Box, \{f\})$$

$$\delta(q, \Box) = (f, 1, N)$$

Beispiel 4.8 (Unär +1)

$$M = (\{q, f\}, \{1\}, \{1, \Box\}, \delta, q, \Box, \{f\})$$

$$\begin{aligned} \delta(q, \Box) &= (f, 1, N) \\ \delta(q, 1) &= (q, 1, R) \end{aligned}$$

Beispiel 4.8 (Unär +1)

$$M = (\{q, f\}, \{1\}, \{1, \Box\}, \delta, q, \Box, \{f\})$$

$$\begin{aligned} \delta(q, \Box) &= (f, 1, N) \\ \delta(q, 1) &= (q, 1, R) \end{aligned}$$

- Beispiellauf:

$$(\epsilon, q, 11) \rightarrow_M$$

Beispiel 4.8 (Unär +1)

$$M = (\{q, f\}, \{1\}, \{1, \Box\}, \delta, q, \Box, \{f\})$$

$$\begin{aligned} \delta(q, \Box) &= (f, 1, N) \\ \delta(q, 1) &= (q, 1, R) \end{aligned}$$

- Beispiellauf:

$$(\epsilon, q, 11) \rightarrow_M (1, q, 1) \rightarrow_M$$

Beispiel 4.8 (Unär +1)

$$M = (\{q, f\}, \{1\}, \{1, \Box\}, \delta, q, \Box, \{f\})$$

$$\begin{aligned} \delta(q, \Box) &= (f, 1, N) \\ \delta(q, 1) &= (q, 1, R) \end{aligned}$$

- Beispiellauf:

$$(\epsilon, q, 11) \to_M (1, q, 1) \to_M (11, q, \epsilon) \to_M$$

Beispiel 4.8 (Unär +1)

$$M = (\{q, f\}, \{1\}, \{1, \Box\}, \delta, q, \Box, \{f\})$$

$$\begin{aligned} \delta(q, \Box) &= (f, 1, N) \\ \delta(q, 1) &= (q, 1, R) \end{aligned}$$

- Beispiellauf:

$$(\epsilon, q, 11) \to_M (1, q, 1) \to_M (11, q, \epsilon) \to_M (11, f, 1)$$

Beispiel 4.8 (Unär +1)

$$M = (\{q, f\}, \{1\}, \{1, \Box\}, \delta, q, \Box, \{f\})$$

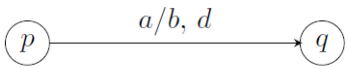
$$\begin{aligned} \delta(q, \Box) &= (f, 1, N) \\ \delta(q, 1) &= (q, 1, R) \end{aligned}$$

- Beispiellauf:

$$(\epsilon, q, 11) \to_M (1, q, 1) \to_M (11, q, \epsilon) \to_M (11, f, 1)$$

- Welche Eingaben führen von q nach f?

Eine graphische Notation für $\delta(p, a) = (q, b, d)$:



Beispiel 4.9 (Binär +1)

ZB $1011 \mapsto 1100$

Beispiel 4.9 (Binär +1)

ZB $1011 \mapsto 1100$

$$M = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \{0,1,\square\}, \delta, q_0, \square, \{q_f\})$$

wobei

$$\begin{array}{lll} \delta(q_0,0) = (q_0,0,R) & \delta(q_1,1) = (q_1,0,L) & \delta(q_2,0) = (q_2,0,L) \\ \delta(q_0,1) = (q_0,1,R) & \delta(q_1,0) = (q_2,1,L) & \delta(q_2,1) = (q_2,1,L) \\ \delta(q_0,\square) = (q_1,\square,L) & \delta(q_1,\square) = (q_f,1,N) & \delta(q_2,\square) = (q_f,\square,R) \end{array}$$

Beispiel 4.9 (Binär +1)

ZB $1011 \mapsto 1100$

$$M = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \{0,1,\square\}, \delta, q_0, \square, \{q_f\})$$

wobei

$$\begin{array}{lll} \delta(q_0,0) = (q_0,0,R) & \delta(q_1,1) = (q_1,0,L) & \delta(q_2,0) = (q_2,0,L) \\ \delta(q_0,1) = (q_0,1,R) & \delta(q_1,0) = (q_2,1,L) & \delta(q_2,1) = (q_2,1,L) \\ \delta(q_0,\square) = (q_1,\square,L) & \delta(q_1,\square) = (q_f,1,N) & \delta(q_2,\square) = (q_f,\square,R) \end{array}$$

Beispiellauf:

$(\epsilon, q_0, 101) \to_M (1, q_0, 01) \to_M (10, q_0, 1) \to_M (101, q_0, \epsilon) \to_M$
$(10, q_1, 1\square) \to_M (1, q_1, 00\square) \to_M$
$(\epsilon, q_2, 110\square) \to_M (\epsilon, q_2, \square 110\square) \to_M$
$(\square, q_f, 110\square)$

Beispiel 4.9 (Binär +1)

ZB $1011 \mapsto 1100$

Die Berechnung der TM M wird als Relation $\to_M$ auf Konfigurationen formalisiert. Falls $\delta(q, \mathit{first}(\beta)) = (q', c, d)$:

$$(\alpha, q, \beta) \to_M \begin{cases} (\alpha, q', c\ \mathit{rest}(\beta)) & \text{falls } d = N \\ (\alpha c, q', \mathit{rest}(\beta)) & \text{falls } d = R \\ (\mathit{butlast}(\alpha), q', \mathit{last}(\alpha)\ c\ \mathit{rest}(\beta)) & \text{falls } d = L \end{cases}$$

wobei

$$\begin{array}{ll} \mathit{first}(aw) = a & \mathit{first}(\epsilon) = \square \\ \mathit{rest}(aw) = w & \mathit{rest}(\epsilon) = \epsilon \\ \mathit{last}(wa) = a & \mathit{last}(\epsilon) = \square \\ \mathit{butlast}(wa) = w & \mathit{butlast}(\epsilon) = \epsilon \end{array}$$

für $a \in \Gamma$ und $w \in \Gamma^*$.

Falls M nichtdeterministisch ist: $\delta(q, \mathit{first}(\beta)) \ni (q', c, d)$

Definition 4.10

Eine Turingmaschine M akzeptiert die Sprache

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F, \alpha, \beta \in \Gamma^*.\ (\epsilon, q_0, w) \to_M^* (\alpha, q, \beta)\}$$

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt Turing-berechenbar gdw es eine Turingmaschine M gibt, so dass für alle $n_1, \ldots n_k, m \in \mathbb{N}$ gilt

$$\begin{array}{l} f(n_1, \ldots, n_k) = m \quad \Leftrightarrow \\ \exists r \in F.\ (\epsilon, q_0, \mathit{bin}(n_1)\#\mathit{bin}(n_2)\#\ldots\#\mathit{bin}(n_k)) \\ \qquad \to_M^* (\square \ldots \square, r, \mathit{bin}(m)\square \ldots \square) \end{array}$$

wobei $\mathit{bin}(n)$ die Binärdarstellung der Zahl n ist.

Beispiel 4.9 (Binär +1)

ZB $1011 \mapsto 1100$

$$M = (\{q_0, q_1, q_2, q_f\}, \{0, 1\}, \{0, 1, \square\}, \delta, q_0, \square, \{q_f\})$$

wobei

$$\begin{array}{lll} \delta(q_0, 0) = (q_0, 0, R) & \delta(q_1, 1) = (q_1, 0, L) & \delta(q_2, 0) = (q_2, 0, L) \\ \delta(q_0, 1) = (q_0, 1, R) & \delta(q_1, 0) = (q_2, 1, L) & \delta(q_2, 1) = (q_2, 1, L) \\ \delta(q_0, \square) = (q_1, \square, L) & \delta(q_1, \square) = (q_f, 1, N) & \delta(q_2, \square) = (q_f, \square, R) \end{array}$$

Beispiellauf:

$$\begin{array}{l} (\epsilon, q_0, 101) \to_M (1, q_0, 01) \to_M (10, q_0, 1) \to_M (101, q_0, \epsilon) \to_M \\ (10, q_1, 1\square) \to_M (1, q_1, 00\square) \to_M \\ (\epsilon, q_2, 110\square) \to_M (\epsilon, q_2, \square 110\square) \to_M \\ (\square, q_f, 110\square) \end{array}$$