5. Berechenbarkeit, Entscheidbarkeit

5. Berechenbarkeit, Entscheidbarkeit

Überblick:

- Was kann man berechnen? Dh:
  - Welche Funktionen kann man in endlicher Zeit berechnen?
  - Welche Eigenschaften von Objekten können in endlicher Zeit entschieden werden?
- Mit welchen Sprachen/Maschinen?
- Was kann man in polynomieller Zeit berechnen?

5.1 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?

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 5.1

Jeder Algorithmus berechnet eine partielle Funktion.
Der Algorithmus

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

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

Beispiel 5.2

Ist die Funktion

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

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

Beispiel 5.2

Ist die Funktion

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

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

Ja, denn es Algorithmen gibt, die π iterativ auf beliebig viele Dezimalstellen genau berechnet werden.

Beispiel 5.3

Ist die Funktion

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

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

Beispiel 5.3

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!

Beispiel 5.3

Ist die Funktion

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

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 5.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls } \underbrace{00\ldots00}_{10^9\,\text{Nullen}} \\ & \text{in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Beispiel 5.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls } \underbrace{00\ldots00}_{10^9\,\text{Nullen}} \\ & \text{in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn f_3 ist entweder die konstante Funktion, die 1 für alle $n \geq 0$ zurückgibt, oder die konstante Funktion, die 0 für alle $n \geq 0$ zurückgibt. Beide sind berechenbar.

Beispiel 5.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls } \underbrace{00\ldots00}_{10^9\,\text{Nullen}} \\ & \text{in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn f_3 ist entweder die konstante Funktion, die 1 für alle $n \geq 0$ zurückgibt, oder die konstante Funktion, die 0 für alle $n \geq 0$ zurückgibt. Beide sind berechenbar.

Dies ist ein nicht-konstruktiver Beweis:
Wir wissen, es gibt einen Algorithmus, der f_3 berechnet,
aber wir wissen nicht, welcher.

Beispiel 5.5

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Beispiel 5.5

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

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.

Beispiel 5.5

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

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.

Wieder ein nicht-konstruktiver Beweis.

Satz 5.6

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

Satz 5.6

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

Beweis:

Es gibt nur abzählbar viele Algorithmen, aber überabzählbar viele Funktionen in $\mathbb{N} \to \{0, 1\}$. $\square$

Erinnerung: Eine Menge M ist abzählbar falls es eine injektive Funktion $M \to \mathbb{N}$ gibt.

Äquivalente Definitionen :

- Entweder gibt es eine Bijektion $M \to \{0, \ldots, n\}$ für ein $n \in \mathbb{N}$, oder eine Bijektion $M \to \mathbb{N}$.
- Es gibt eine Nummerierung der Elemente von M.

Eine Menge ist überabzählbar wenn sie nicht abzählbar ist.

Lemma 5.7

Σ^* ist abzählbar (falls Σ endlich).

Lemma 5.7

Σ^* ist abzählbar (falls Σ endlich).

Beweis:

Durch Beispiel:

$$\begin{array}{lcl} \{0,1\}^* & = & \{\epsilon, \quad 0, \quad 1, \quad 00, \quad 01, \quad 10, \quad 11, \quad 000, \ldots\} \\ \mathbb{N} & = & \{0, \quad 1, \quad 2, \quad 3, \quad 4, \quad 5, \quad 6, \quad 7, \ldots\} \end{array}$$

$\square$

Lemma 5.7

Σ^* ist abzählbar (falls Σ endlich).

Beweis:

Durch Beispiel:

$$\begin{array}{lcl} \{0,1\}^* & = & \{\epsilon, \quad 0, \quad 1, \quad 00, \quad 01, \quad 10, \quad 11, \quad 000, \ldots\} \\ \mathbb{N} & = & \{0, \quad 1, \quad 2, \quad 3, \quad 4, \quad 5, \quad 6, \quad 7, \ldots\} \end{array}$$

$\square$

Es folgt: Wenn Algorithmen als Wörter über ein endliches Alphabet sich kodieren lassen, dann ist die Menge der Algorithmen abzählbar.
Dies Eingeschaft gilt für alle existierenden Formalismen für die Darstellung von Algorithmen (und insbesondere für alle Programmiersprachen).

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$ *ist überabzählbar.*

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$ *ist überabzählbar.*

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

f_0	
f_1	
f_2	
f_3	
$\vdots$	

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$ *ist überabzählbar.*

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	$\ldots$
f_0					
f_1					
f_2					
f_3					
$\vdots$					

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$ *ist überabzählbar.*

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	$\ldots$
f_0	0	1	1	0	$\ldots$
f_1					
f_2					
f_3					
$\vdots$					

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$ *ist überabzählbar.*

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	$\ldots$
f_0	0	1	1	0	$\ldots$
f_1	1	0	0	0	$\ldots$
f_2					
f_3					
$\vdots$					

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$ *ist überabzählbar.*

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	$\ldots$
f_0	0	1	1	0	$\ldots$
f_1	1	0	0	0	$\ldots$
f_2	0	0	1	0	$\ldots$
f_3					
$\vdots$					

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$ *ist überabzählbar.*

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	$\ldots$
f_0	0	1	1	0	$\ldots$
f_1	1	0	0	0	$\ldots$
f_2	0	0	1	0	$\ldots$
f_3	0	0	1	0	$\ldots$
$\vdots$					

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$ *ist überabzählbar.*

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	$\dots$
f_0	0	1	1	0	$\dots$
f_1	1	0	0	0	$\dots$
f_2	0	0	1	0	$\dots$
f_3	0	0	1	0	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$ *ist überabzählbar.*

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	$\dots$
f_0	0	1	1	0	$\dots$
f_1	1	0	0	0	$\dots$
f_2	0	0	1	0	$\dots$
f_3	0	0	1	0	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Eine Funktion f, die nicht in der Tabelle ist: $\neg$Diagonale!

f	1	1	0	1	$\dots$

Widerspruch!

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	$\dots$
f_0	0	1	1	0	$\dots$
f_1	1	0	0	0	$\dots$
f_2	0	0	1	0	$\dots$
f_3	0	0	1	0	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Eine Funktion f, die nicht in der Tabelle ist: $\neg$Diagonale!

f	1	1	0	1	$\dots$

Widerspruch!

Formal: Sei $f(i) := 1 - f_i(i)$. Dann $f \neq f_i$ für alle i, da $f(i) \neq f_i(i)$. $\square$

Satz 5.9

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es unberechenbare Funktionen $\mathbb{N} \to \{0, 1\}$.

Satz 5.9

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es unberechenbare Funktionen $\mathbb{N} \to \{0, 1\}$.

Beweis:

Sei $\mathcal{F}$ die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$.
Sei $\mathcal{A}$ die Menge der Wörter, die Algorithmen kodieren.
Beweis durch Widerspruch.

Satz 5.9

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es unberechenbare Funktionen $\mathbb{N} \to \{0, 1\}$.

Beweis:

Sei $\mathcal{F}$ die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$.
Sei $\mathcal{A}$ die Menge der Wörter, die Algorithmen kodieren.
Beweis durch Widerspruch.
Annahme: Alle Funktionen von $\mathcal{F}$ sind berechenbar.

Satz 5.9

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es unberechenbare Funktionen $\mathbb{N} \to \{0, 1\}$.

Beweis:

Sei $\mathcal{F}$ die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$.
Sei $\mathcal{A}$ die Menge der Wörter, die Algorithmen kodieren.
Beweis durch Widerspruch.
Annahme: Alle Funktionen von $\mathcal{F}$ sind berechenbar.
Da $\mathcal{A}$ abzählbar ist (Lemma 5.7), gibt es eine injektive Funktion $anum\colon \mathcal{A} \to \mathbb{N}$.
Definiere $fnum\colon \mathcal{F} \to \mathbb{N}$ durch

$$fnum(f) = \min\{anum(a) \mid a \text{ berechnet } f\ \}$$

Satz 5.9

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es unberechenbare Funktionen $\mathbb{N} \to \{0, 1\}$.

Beweis:

Sei $\mathcal{F}$ die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$.
Sei $\mathcal{A}$ die Menge der Wörter, die Algorithmen kodieren.
Beweis durch Widerspruch.
Annahme: Alle Funktionen von $\mathcal{F}$ sind berechenbar.
Da $\mathcal{A}$ abzählbar ist (Lemma 5.7), gibt es eine injektive Funktion $anum\colon \mathcal{A} \to \mathbb{N}$.
Definiere $fnum\colon \mathcal{F} \to \mathbb{N}$ durch

$$fnum(f) = \min\{anum(a) \mid a \text{ berechnet } f\ \}$$

$fnum$ is injektiv: Wenn $fnum(f_1) = fnum(f_2)$ dann gibt es einen Algorithmus der sowohl f_1 wie auch f_2 berechnet und so $f_1 = f_2$.
Aber dann ist $\mathcal{F}$ abzählbar. ` zu Satz 5.8 $\square$

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
- ...

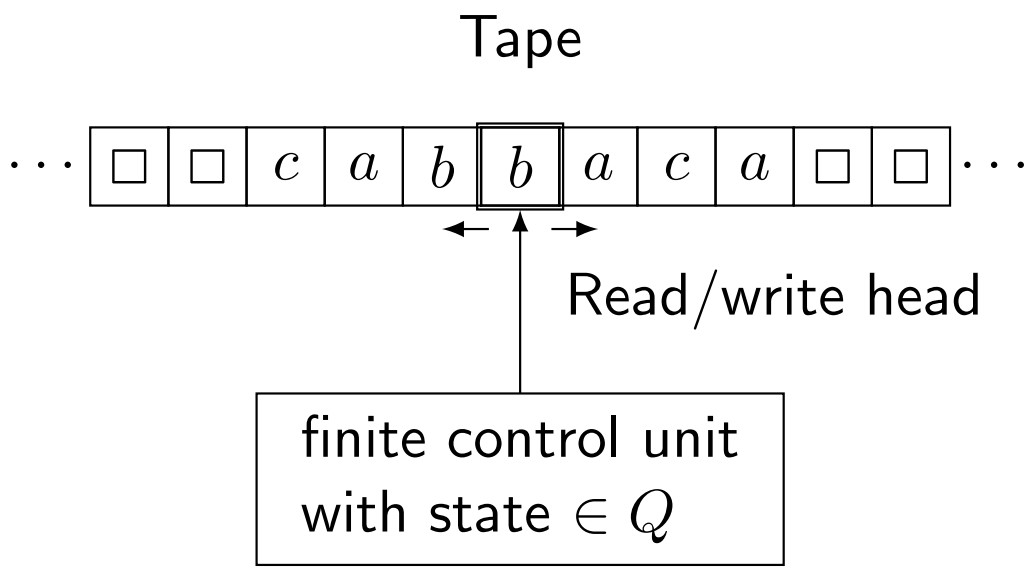
Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

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

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.

Die Church-Turing-These ist keine formale Aussage und so nicht beweisbar. Sie wird jedoch allgemein akzeptiert.


Tape
$\cdots$ $\square$ $\square$ c a b b a c a $\square$ $\square$ $\cdots$
Read/write head
finite control unit
with state $\in Q$

Definition 5.10

Eine Turingmaschine (TM) ist ein 7-Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ 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 partiell sein.
- $q_0 \in Q$ ist der Startzustand.
- $\Box \in \Gamma \setminus \Sigma$ ist das Leerzeichen.
- $F \subseteq Q$ ist die Menge der akzeptierenden oder Endzustände.

Definition 5.10

Eine Turingmaschine (TM) ist ein 7-Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ 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 partiell sein.
- $q_0 \in Q$ ist der Startzustand.
- $\Box \in \Gamma \setminus \Sigma$ ist das Leerzeichen.
- $F \subseteq Q$ ist die Menge der akzeptierenden oder Endzustände.

Annahme: $\delta(q, a)$ is nicht definiert für alle $q \in F$ und $a \in \Gamma$.

Definition 5.10

Eine Turingmaschine (TM) ist ein 7-Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ 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 partiell sein.
- $q_0 \in Q$ ist der Startzustand.
- $\Box \in \Gamma \setminus \Sigma$ ist das Leerzeichen.
- $F \subseteq Q$ ist die Menge der akzeptierenden oder Endzustände.

Annahme: $\delta(q, a)$ is nicht definiert für alle $q \in F$ und $a \in \Gamma$.

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

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

- 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 5.11

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 5.11

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 $\rightarrow_M$ auf Konfigurationen formalisiert. Falls $\delta(q, \mathit{first}(\beta)) = (q', c, D)$:

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

wobei

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

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

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

Beispiel 5.12 (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 5.12 (Unär +1)

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

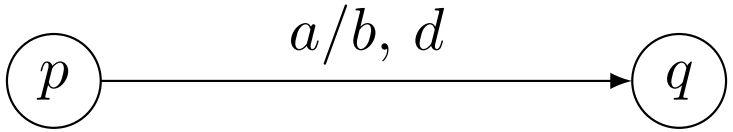
$$\begin{aligned} \delta(q, \Box) &= (f, 1, N) \\ \delta(q, 1) &= (q, 1, R) \end{aligned}$$

- Beispiellauf:

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

- Welche Eingaben führen von q nach f?

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



Beispiel 5.13 (Binär $+1$)

ZB $1011 \mapsto 1100$

Beispiel 5.13 (Binär +1)

ZB $1011 \mapsto 1100$

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

wobei

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

Beispiel 5.13 (Binär +1)

ZB $1011 \mapsto 1100$

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

wobei

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

Beispiellauf:

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

Definition 5.14

Eine Turingmaschine M akzeptiert die Sprache

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

Definition 5.14

Eine Turingmaschine M akzeptiert die Sprache

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

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

$$\begin{aligned} & f(n_1, \ldots, n_k) = m \quad \Leftrightarrow \\ & \exists r \in F.\ (\epsilon, q_0, bin(n_1)\#bin(n_2)\# \ldots \#bin(n_k)) \\ & \qquad \rightarrow_M^* (\Box \ldots \Box, r, bin(m)\Box \ldots \Box) \end{aligned}$$

wobei $bin(n)$ die Binärdarstellung der Zahl n ist.

Definition 5.14

Eine Turingmaschine M akzeptiert die Sprache

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

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

$$\begin{aligned} & f(n_1, \ldots, n_k) = m \quad \Leftrightarrow \\ & \exists r \in F.\ (\epsilon, q_0, bin(n_1)\#bin(n_2)\# \ldots \#bin(n_k)) \\ & \qquad \rightarrow_M^* (\Box \ldots \Box, r, bin(m)\Box \ldots \Box) \end{aligned}$$

wobei $bin(n)$ die Binärdarstellung der Zahl n ist.

Eine Funktion $f : \Sigma^* \rightarrow \Sigma^*$ heißt Turing-berechenbar gdw es eine Turingmaschine M gibt, so dass für alle $u, v \in \Sigma^*$ gilt

$$f(u) = v \quad \Leftrightarrow \quad \exists r \in F.\ (\epsilon, q_0, u) \rightarrow_M^* (\Box \ldots \Box, r, v\Box \ldots \Box)$$

Zum Halten/Terminieren von TM

Zum Halten/Terminieren von TM

Eine TM hält wenn sie eine Konfiguration $(\alpha, q, a\beta)$ erreicht und $\delta(q, a)$ nicht definiert oder (bei einer nichtdetermistischen TM) $\delta(q, a) = \emptyset$.

Zum Halten/Terminieren von TM

Eine TM hält wenn sie eine Konfiguration $(\alpha, q, a\beta)$ erreicht und $\delta(q, a)$ nicht definiert oder (bei einer nichtdetermistischen TM) $\delta(q, a) = \emptyset$.

Nach Annahme hält eine TM immer, wenn sie einen Endzustand erreicht. Damit ist die von einer TM berechnete Funktion wohldefiniert.

Zum Halten/Terminieren von TM

Eine TM hält wenn sie eine Konfiguration $(\alpha, q, a\beta)$ erreicht und $\delta(q, a)$ nicht definiert oder (bei einer nichtdetermistischen TM) $\delta(q, a) = \emptyset$.

Nach Annahme hält eine TM immer, wenn sie einen Endzustand erreicht. Damit ist die von einer TM berechnete Funktion wohldefiniert.

Eine TM kann auch halten, bevor sie einen Endzustand erreicht.

Satz 5.15

Zu jeder nichtdeterministischen TM N gibt es eine deterministische TM M mit $L(N) = L(M)$.

Satz 5.15

Zu jeder nichtdeterministischen TM N gibt es eine deterministische TM M mit $L(N) = L(M)$.

Beweis:

M durchsucht den Baum der Berechnungen von N in *Breitensuche*, beginnend mit der Startkonfiguration (ϵ, q_0, w), eine Ebene nach der anderen.

Satz 5.15

Zu jeder nichtdeterministischen TM N gibt es eine deterministische TM M mit $L(N) = L(M)$.

Beweis:

M durchsucht den Baum der Berechnungen von N in *Breitensuche*, beginnend mit der Startkonfiguration (ϵ, q_0, w), eine Ebene nach der anderen.
Gibt es in dem Baum (auf Ebene n) eine Konfigurationen mit Endzustand, so wird diese (nach Zeit $O(c^n)$) gefunden. $\square$

Satz 5.16

Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierarchie.

Satz 5.16

Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierarchie.

Beweis:

Wir beschreiben nur die Beweisidee. (Mehr Details: [Schöning]).

Satz 5.16

Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierarchie.

Beweis:

Wir beschreiben nur die Beweisidee. (Mehr Details: [Schöning]).

„$\Longrightarrow$“: Grammatikregeln können direkt die Rechenregeln einer TM simulieren.

Satz 5.16

Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierarchie.

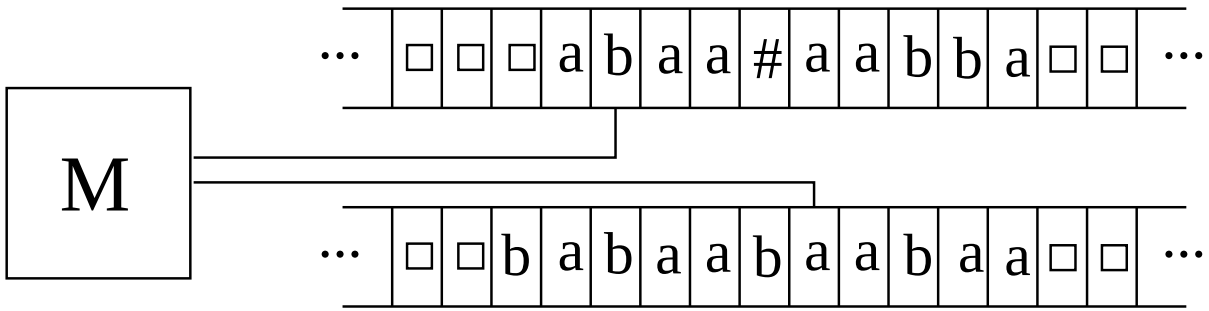
Beweis:

Wir beschreiben nur die Beweisidee. (Mehr Details: [Schöning]).

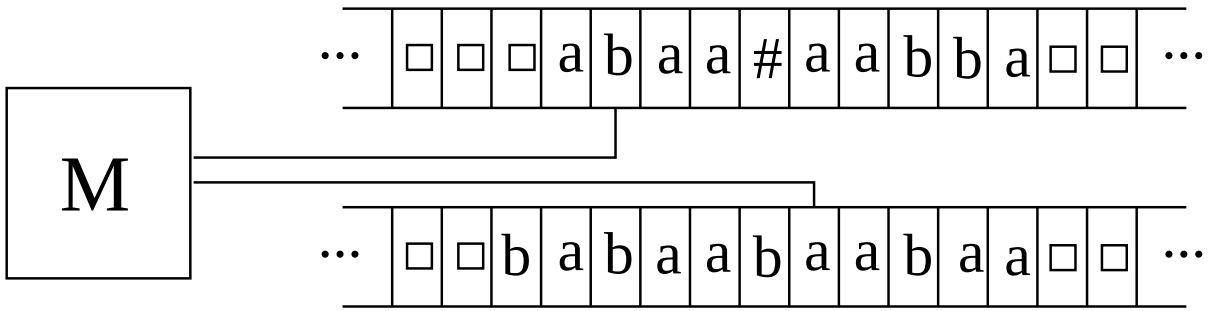
„$\Longrightarrow$“: Grammatikregeln können direkt die Rechenregeln einer TM simulieren.

„$\Longleftarrow$“: Die (nichtdeterministische) TM versucht von ihrer Eingabe aus das Startsymbol der Grammatik zu erreichen, indem sie die Produktionen der Grammatik von rechts nach links anwendet, (nichtdeterministisch) an jeder möglichen Stelle. $\square$

Eine beliebte Modellvariante ist die k-Band-Turingmaschine:



Eine beliebte Modellvariante ist die k-Band-Turingmaschine:



Die k Köpfe sind völlig unabhängig voneinander:

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, N\}^k$$

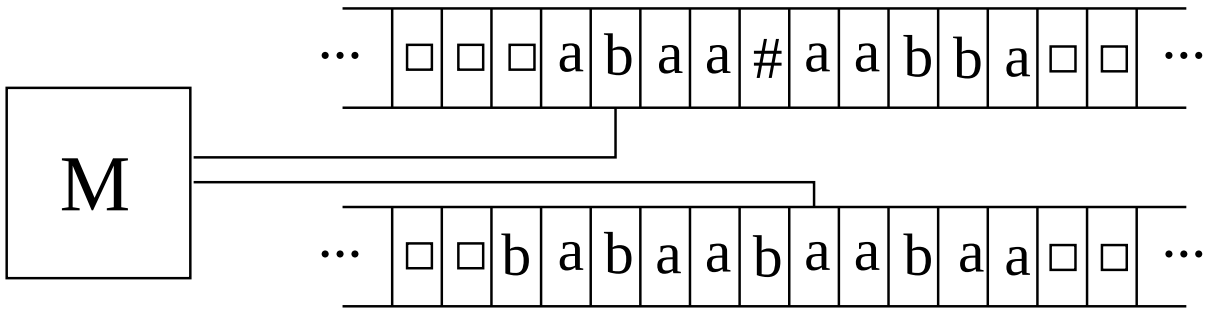
Satz 5.17

Jede k-Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

Satz 5.17

Jede k-Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

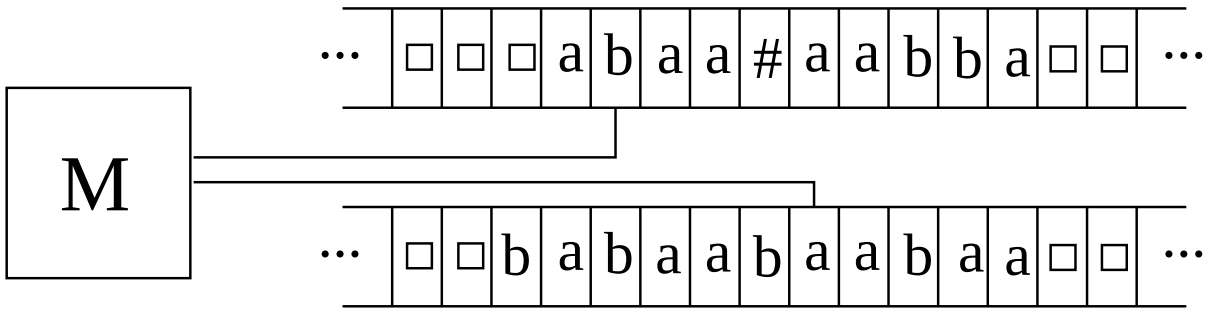
Beweisidee: Aus



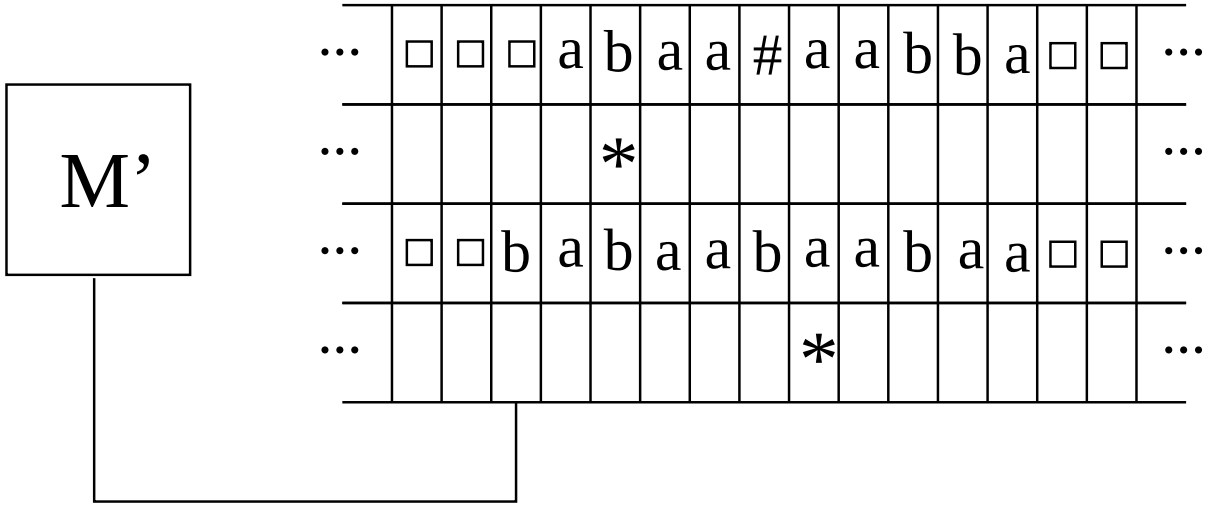
Satz 5.17

Jede k-Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

Beweisidee: Aus



wird



Beweisskizze:

- $\Gamma' := (\Gamma \times \{\star, \square\})^k$
- M' simuliert einen M-Schritt durch mehrere Schritte: M'
  - startet mit Kopf links von allen $\star$.
  - geht nach rechts bis alle $\star$ überschritten sind,
    und merkt sich dabei (in Q') die Zeichen über jedem $\star$.
  - hat jetzt alle Information, um δ_M anzuwenden.
  - geht nach links über alle $\star$ hinweg und führt dabei δ_M aus.

Beweisskizze:

- $\Gamma' := (\Gamma \times \{\star, \square\})^k$
- M' simuliert einen M-Schritt durch mehrere Schritte: M'
  - startet mit Kopf links von allen $\star$.
  - geht nach rechts bis alle $\star$ überschritten sind, und merkt sich dabei (in Q') die Zeichen über jedem $\star$.
  - hat jetzt alle Information, um δ_M anzuwenden.
  - geht nach links über alle $\star$ hinweg und führt dabei δ_M aus.

Beobachtung:

n Schritte von M lassens sich durch
$O(n^2)$ Schritte von M' simulieren.

Denn nach n Schritten von M trennen $\leq 2n$ Felder linkesten und rechtesten Kopf. Obige Simulation eines M-Schritts braucht daher $O(n)$ M'-Schritte. Simulation von n Schritten: $O(n^2)$ Schritte.

Die folgenden Basismaschinen sind leicht programmierbar:

- Band i := Band $i + 1$
- Band i := Band $i - 1$
- Band i := 0
- Band i := Band j

Seien $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, q_i, \Box, F_i)$, $i = 1, 2$.
Die sequentielle Komposition (Hintereinanderschaltung) von M_1 und M_2 bezeichnen wir mit

$$\longrightarrow M_1 \longrightarrow M_2 \longrightarrow$$

Sie ist wie folgt definiert:

$$M := (Q_1 \cup Q_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, q_1, \Box, F_2)$$

wobei (oE) $Q_1 \cap Q_2 = \emptyset$ und

$$\delta := \delta_1 \cup \delta_2 \cup \{(f_1, a) \mapsto (q_2, a, N) \mid f_1 \in F_1, a \in \Gamma_1\}$$

Sind f_1 und f_2 Endzustände von M so bezeichnet

$$\begin{array}{ccccc} \longrightarrow & M & \xrightarrow{f_1} & M_1 & \longrightarrow \\ & \downarrow f_2 & & & \\ & M_2 & & & \\ & \downarrow & & & \end{array}$$

eine Fallunterscheidung, dh eine TM, die vom Endzustand f_1 von M nach M_1 übergeht, und von f_2 aus nach M_2.

Sind f_1 und f_2 Endzustände von M so bezeichnet

$$\begin{array}{ccccc} \longrightarrow & M & \xrightarrow{f_1} & M_1 & \longrightarrow \\ & \downarrow f_2 & & & \\ & M_2 & & & \\ & \downarrow & & & \end{array}$$

eine Fallunterscheidung, dh eine TM, die vom Endzustand f_1 von M nach M_1 übergeht, und von f_2 aus nach M_2.

Die folgende TM nennen wir „Band=0?“ (bzw „Band $i = 0$?“):

$$\begin{array}{rcll} \delta(q_0, 0) & = & (q_0, 0, R) & \\ \delta(q_0, \Box) & = & (ja, \Box, L) & \\ \delta(q_0, a) & = & (nein, a, N) & \quad \text{für } a \neq 0, \Box \end{array}$$

wobei ja und $nein$ Endzustände sind.

Analog zur Fallunterscheidung kann man auch eine TM für eine Schleife konstruieren

$$\longrightarrow \quad \begin{array}{l} \text{Band } i = 0? \quad \xrightarrow{ja} \\ \uparrow \downarrow nein \\ M \end{array}$$

die sich wie `while` Band $i \neq 0$ `do` M verhält.

Analog zur Fallunterscheidung kann man auch eine TM für eine Schleife konstruieren

$$\longrightarrow \quad \text{Band } i = 0? \quad \xrightarrow{ja}$$
$$\uparrow \downarrow nein$$
$$M$$

die sich wie `while` Band $i \neq 0$ `do` M verhält.

Moral: Mit TM kann man imperativ programmieren:

```
:=
;
if
while
```

WHILE	$\equiv$	strukturierte Programme mit `while`-Schleifen
GOTO	$\equiv$	Assembler

Wir definieren WHILE- und GOTO-Berechenbarkeit und zeigen ihre Äquivalenz mit Turing-Berechenbarkeit.

WHILE	$\equiv$	strukturierte Programme mit `while`-Schleifen
GOTO	$\equiv$	Assembler

Wir definieren WHILE- und GOTO-Berechenbarkeit und zeigen ihre Äquivalenz mit Turing-Berechenbarkeit.

Syntax von WHILE-Programmen:

$$
\begin{array}{rcl}
P & \rightarrow & X \texttt{ := } X \texttt{ + } C \\
 & | & X \texttt{ := } X \texttt{ - } C \\
 & | & P \texttt{ ; } P \\
 & | & \texttt{IF } X = 0 \texttt{ DO } P \texttt{ ELSE } Q \texttt{ END} \\
 & | & \texttt{WHILE } X \neq 0 \texttt{ DO } P \texttt{ END}
\end{array}
$$

wobei X eine der Variablen $x_0, x_1, \ldots$ und C eine der Konstanten $0, 1, \ldots$ sein kann.

WHILE	$\equiv$	strukturierte Programme mit `while`-Schleifen
GOTO	$\equiv$	Assembler

Wir definieren WHILE- und GOTO-Berechenbarkeit und zeigen ihre Äquivalenz mit Turing-Berechenbarkeit.

Syntax von WHILE-Programmen:

$$
\begin{array}{rcl}
P & \rightarrow & X \texttt{ := } X \texttt{ + } C \\
 & | & X \texttt{ := } X \texttt{ - } C \\
 & | & P \texttt{ ; } P \\
 & | & \texttt{IF } X = 0 \texttt{ DO } P \texttt{ ELSE } Q \texttt{ END} \\
 & | & \texttt{WHILE } X \neq 0 \texttt{ DO } P \texttt{ END}
\end{array}
$$

wobei X eine der Variablen $x_0, x_1, \ldots$ und C eine der Konstanten $0, 1, \ldots$ sein kann.

Beispiel 5.18

$\texttt{WHILE } x_2 \neq 0 \texttt{ DO } x_1 \texttt{ := } x_0\texttt{+1 END}$

Die modifizierte Differenz ist $m \dot{-} n \;:=\; \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

x_i `:=` x_j `+` n	Neuer Wert von x_i ist $x_j + n$.
x_i `:=` x_j `-` n	Neuer Wert von x_i ist $x_j \dot{-} n$.
P_1 ; P_2	Führe zuerst P_1 und dann P_2 aus.
`WHILE` $x_i \neq 0$ `DO` P `END`	Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Die modifizierte Differenz ist $m \dot{-} n \;:=\; \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

x_i `:=` x_j `+` n	Neuer Wert von x_i ist $x_j + n$.
x_i `:=` x_j `-` n	Neuer Wert von x_i ist $x_j \dot{-} n$.
P_1 ; P_2	Führe zuerst P_1 und dann P_2 aus.
`WHILE` $x_i \neq 0$ `DO` P `END`	Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Beispiel 5.19

`WHILE` $x_1 \neq 0$ `DO` x_2 `:=` x_2 `+` 1; $x_1 := x_1$ `-` 1 `END`
simuliert
x_2 `:=` x_2 `+` x_1

Die modifizierte Differenz ist $m \dot{-} n \;:=\; \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

x_i `:=` x_j `+` n	Neuer Wert von x_i ist $x_j + n$.
x_i `:=` x_j `-` n	Neuer Wert von x_i ist $x_j \dot{-} n$.
P_1 ; P_2	Führe zuerst P_1 und dann P_2 aus.
`WHILE` $x_i \neq 0$ `DO` P `END`	Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Beispiel 5.19

`WHILE` $x_1 \neq 0$ `DO` x_2 `:=` x_2 `+` 1; $x_1 := x_1$ `-` 1 `END`
simuliert
x_2 `:=` x_2 `+` x_1

Zu Beginn der Ausführung stehen die Eingaben in $x_1, \ldots, x_k$.
Alle anderen Variablen sind 0. Die Ausgabe wird in x_0 berechnet.

Syntaktische Abkürzungen („Zucker“):

x_i `:=` x_j $\equiv$ x_i `:=` x_j `+` 0

x_i `:=` n $\equiv$ x_i `:=` x_j `+` n

(wobei an x_j nirgends zugewiesen wird)

x_i `:=` x_j `+` x_k $\equiv$ x_i `:=` x_j`;`

(mit $i \neq k$)

```
WHILE x_k ≠ 0 DO
   x_i := x_i + 1; x_k := x_k - 1
END
```

x_i `:=` x_j `*` x_k $\equiv$ x_i `:=` 0`;`

(mit $i \neq j, k$)

```
WHILE x_k ≠ 0 DO
   x_i := x_i + x_j; x_k := x_k - 1
END
```

`DIV`, `MOD`, `IF` $x = 0$ `THEN` P `END` ...

Definition 5.20

Eine totale Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.) terminiert mit $f(n_1, \ldots, n_k)$ in x_0.

Definition 5.20

Eine totale Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.) terminiert mit $f(n_1, \ldots, n_k)$ in x_0.

Definition 5.21

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.)

- terminiert mit $f(n_1, \ldots, n_k)$ in x_0,
  falls $f(n_1, \ldots, n_k)$ definiert ist,
- terminiert nicht, falls $f(n_1, \ldots, n_k)$ undefiniert ist.

Turingmaschinen können WHILE-Programme simulieren:

Satz 5.22 (WHILE $\rightarrow$ TM)

Jede WHILE-berechenbare Funktion ist auch Turing-berechenbar.

Beweis:

Jede Programmvariable wird auf einem eigenen Band gespeichert. Wir haben bereits gezeigt: Alle Konstrukte der WHILE-Sprache können von einer Mehrband-TM simuliert werden, und eine Mehrband-TM kann von einer 1-Band TM simuliert werden. $\square$

GOTO TM

WHILE

Übersetzungen

Ein GOTO-Programm ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Mögliche Anweisungen A_i sind:

x_i `:=` x_j `+` n
x_i `:=` x_j `-` n
`GOTO` M_i
`IF` $x_i = n$ `GOTO` M_j
`HALT`

Die Semantik ist wie erwartet.

Ein GOTO-Programm ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Mögliche Anweisungen A_i sind:

$$\begin{array}{l} x_i \ \texttt{:=}\ x_j \ \texttt{+}\ n \\ x_i \ \texttt{:=}\ x_j \ \texttt{-}\ n \\ \texttt{GOTO}\ M_i \\ \texttt{IF}\ x_i = n\ \texttt{GOTO}\ M_j \\ \texttt{HALT} \end{array}$$

Die Semantik ist wie erwartet.

Fakt 5.23 (WHILE $\rightarrow$ GOTO)

Jedes WHILE-Programm kann durch ein GOTO-Programm simuliert werden.

Satz 5.24 (GOTO $\rightarrow$ WHILE)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.

Satz 5.24 (GOTO $\rightarrow$ WHILE)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.

Beweis: Simuliere $M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$ durch

$$
\begin{array}{l}
pc \texttt{ := } 1\texttt{;} \\
\texttt{WHILE } pc \neq 0 \texttt{ DO} \\
\quad \texttt{IF } pc = 1 \texttt{ THEN } P_1 \texttt{ ELSE} \\
\quad \vdots \\
\quad \texttt{IF } pc = k \texttt{ THEN } P_k \texttt{ ELSE } pc := 0 \\
\texttt{END}
\end{array}
$$

wobei $A_i \mapsto P_i$ wie folgt definiert ist:

$$
\begin{array}{lcl}
x_i \texttt{ := } x_j \texttt{ +/- } n & \mapsto & x_i \texttt{ := } x_j \texttt{ +/- } n\texttt{;}\ pc \texttt{ := } pc \texttt{ + } 1 \\
\texttt{GOTO } M_i & \mapsto & pc \texttt{ := } i \\
\texttt{IF } x_j = 0 \texttt{ GOTO } M_i & \mapsto & \texttt{IF } x_j = 0 \\
 & & \texttt{THEN } pc \texttt{ := } i \texttt{ ELSE } pc \texttt{ := } pc \texttt{ + } 1 \texttt{ END} \\
\texttt{HALT} & \mapsto & pc \texttt{ := } 0
\end{array}
$$

$\square$

Korollar 5.25

WHILE- und GOTO-Berechenbarkeit sind äquivalent.

Korollar 5.25

WHILE- und GOTO-Berechenbarkeit sind äquivalent.

Korollar 5.26 (Kleenesche Normalform)

Jedes WHILE-Programm ist zu einem WHILE-Programm mit genau einer WHILE-Schleife äquivalent.

$$\begin{tikzcd}
\text{GOTO} & & \text{TM} \\
& \text{WHILE} \arrow[ul, shift left] \arrow[ul, shift right] \arrow[ur] &
\end{tikzcd}$$

Übersetzungen

Satz 5.27 (TM $\rightarrow$ GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Satz 5.27 (TM $\rightarrow$ GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung: TM $(Q, \Sigma, \Gamma, \delta, q_0, \Box, F) \rightarrow$ GOTO-Programm.

Satz 5.27 (TM $\rightarrow$ GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung: TM $(Q, \Sigma, \Gamma, \delta, q_0, \Box, F) \rightarrow$ GOTO-Programm.

$$Q = \{q_0, \ldots, q_k\} \qquad \Gamma = \{a_0(= \Box), \ldots, a_n\}$$

Satz 5.27 (TM $\rightarrow$ GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung: TM $(Q, \Sigma, \Gamma, \delta, q_0, \Box, F) \rightarrow$ GOTO-Programm.

$$Q = \{q_0, \ldots, q_k\} \qquad \Gamma = \{a_0(= \Box), \ldots, a_n\}$$

Eine Konfiguration

$$(a_{i_p} \ldots a_{i_1},\ q_l,\ a_{j_1} \ldots a_{j_q})$$

wird durch die Programmvariablen x, y, z wie folgt repräsentiert:

$$x = (i_p \ldots i_1)_b, \quad y = (j_q \ldots j_1)_b, \quad z = l$$

wobei $(i_p \ldots i_1)_b$ die Zahl $i_p \ldots i_1$ zur Basis $b := n + 1$ ist:

$$x = \sum_{r=1}^{p} i_r b^{r-1}$$

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

$$
\begin{array}{rl}
M: & \texttt{IF}\ z \in F\ \texttt{GOTO}\ M_{end}\texttt{;} \\
 & a\ \texttt{:=}\ y\ \texttt{MOD}\ b\texttt{;} \\
 & \texttt{IF}\ z = 0\ \texttt{AND}\ a = 0\ \texttt{GOTO}\ M_{00}\texttt{;} \\
 & \texttt{IF}\ z = 0\ \texttt{AND}\ a = 1\ \texttt{GOTO}\ M_{01}\texttt{;} \\
 & \ldots \\
 & \texttt{IF}\ z = k\ \texttt{AND}\ a = n\ \texttt{GOTO}\ M_{kn}\texttt{;} \\
M_{00}: & P_{00}\texttt{;}\ \texttt{GOTO}\ M\texttt{;} \\
M_{01}: & P_{01}\texttt{;}\ \texttt{GOTO}\ M\texttt{;} \\
 & \ldots \\
M_{kn}: & P_{kn}\texttt{;}\ \texttt{GOTO}\ M
\end{array}
$$

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist.

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

$$
\begin{array}{rl}
M: & \texttt{IF}\ z \in F\ \texttt{GOTO}\ M_{end}\texttt{;} \\
 & a\ \texttt{:=}\ y\ \texttt{MOD}\ b\texttt{;} \\
 & \texttt{IF}\ z = 0\ \texttt{AND}\ a = 0\ \texttt{GOTO}\ M_{00}\texttt{;} \\
 & \texttt{IF}\ z = 0\ \texttt{AND}\ a = 1\ \texttt{GOTO}\ M_{01}\texttt{;} \\
 & \ldots \\
 & \texttt{IF}\ z = k\ \texttt{AND}\ a = n\ \texttt{GOTO}\ M_{kn}\texttt{;} \\
M_{00}: & P_{00}\texttt{;}\ \texttt{GOTO}\ M\texttt{;} \\
M_{01}: & P_{01}\texttt{;}\ \texttt{GOTO}\ M\texttt{;} \\
 & \ldots \\
M_{kn}: & P_{kn}\texttt{;}\ \texttt{GOTO}\ M
\end{array}
$$

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist. Für $D = L$:

z `:=` r`;`	Zustand aktualisieren
y `:=` y `DIV` b`;`	Löschen von a_j
y `:=` b`*`y `+` s`;`	Schreiben von a_s
y `:=` b`*`y `+ (`x `MOD` b`);`	Bewegung L (I): Einfügen von a_{i_1} in y
x `:=` x `DIV` b	Bewegung L (II): Löschen von a_{i_1} aus x

$$
\begin{tikzcd}
\text{GOTO} \arrow[dr, shift left] & & \text{TM} \arrow[ll] \\
& \text{WHILE} \arrow[ul, shift left] \arrow[ur] &
\end{tikzcd}
$$

Übersetzungen

5.2 Unentscheidbarkeit des Halteproblems

Definition 5.28

Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt entscheidbar gdw ihre charakteristische Funktion

$$\chi_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

5.2 Unentscheidbarkeit des Halteproblems

Definition 5.28

Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt entscheidbar gdw ihre charakteristische Funktion

$$\chi_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Eine Eigenschaft/Problem $P(x)$ heißt entscheidbar gdw $\{x \mid P(x)\}$ entscheidbar ist.

5.2 Unentscheidbarkeit des Halteproblems

Definition 5.28

Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt entscheidbar gdw ihre charakteristische Funktion

$$\chi_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Eine Eigenschaft/Problem $P(x)$ heißt entscheidbar gdw $\{x \mid P(x)\}$ entscheidbar ist.

Fakt 5.29

- *Ist $A \subseteq \Sigma^*$ entscheidbar, dann gibt es eine TM M mit $L(M) = A$.*
- *Die entscheidbaren Mengen sind abgeschlossen unter Komplement: Ist A entscheidbar, dann auch $\overline{A}$.*

Kodierung einer TM als Wort über $\Sigma = \{0, 1\}$, exemplarisch:

- Sei $\Gamma = \{a_0, \ldots, a_k\}$ und $Q = \{q_0, \ldots, q_n\}$.
- $\delta(q_i, a_j) = (q_{i'}, a_{j'}, d)$ wird kodiert als

$$\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(m)$$

  wobei $bin : \mathbb{N} \to \{0, 1\}^*$ die Binärkodierung einer Zahl ist und $m = 0/1/2$ falls $d = L/R/N$.
- Kodierung von δ: Konkatenation der Kodierungen aller $\delta(., .) = (., ., .)$, in beliebiger Reihenfolge.
- Kodierung von $\{0, 1, \#\}^*$ in $\{0, 1\}^*$:

$$\begin{array}{rcl} 0 & \mapsto & 00 \\ 1 & \mapsto & 01 \\ \# & \mapsto & 11 \end{array}$$

Nicht jedes Wort über $\{0,1\}^*$ kodiert eine TM.

Nicht jedes Wort über $\{0,1\}^*$ kodiert eine TM.

Sei $\hat{M}$ eine beliebige feste TM.

Definition 5.30

Die zu einem Wort $w \in \{0,1\}^*$ gehörige TM M_w ist

$$M_w := \begin{cases} M & \text{falls } w \text{ Kodierung von } M \text{ ist} \\ \hat{M} & \text{sonst} \end{cases}$$

Definition 5.31

$M[w]$ ist Abk. für „Maschine M mit Eingabe w“

$M[w]\downarrow$ bedeutet, dass $M[w]$ terminiert/hält.

Definition 5.31

$M[w]$ ist Abk. für „Maschine M mit Eingabe w“

$M[w]\downarrow$ bedeutet, dass $M[w]$ terminiert/hält.

Definition 5.32 (Spezielles Halteproblem)

Gegeben: Ein Wort $w \in \{0,1\}^*$.

Problem: Hält M_w bei Eingabe w ?

Definition 5.31

$M[w]$ ist Abk. für „Maschine M mit Eingabe w“

$M[w]\downarrow$ bedeutet, dass $M[w]$ terminiert/hält.

Definition 5.32 (Spezielles Halteproblem)

Gegeben: Ein Wort $w \in \{0,1\}^*$.

Problem: Hält M_w bei Eingabe w ?

Als Menge:

$$K := \{w \in \{0,1\}^* \mid M_w[w]\downarrow\}$$

Satz 5.33

Das spezielle Halteproblem ist nicht entscheidbar.

Satz 5.33

Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.

Satz 5.33

Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.
Dann ist auch folgende Funktion f berechenbar:

$$f(w) := \begin{cases} 0 & \text{falls } \chi_K(w) = 0 \\ \perp & \text{falls } \chi_K(w) = 1 \end{cases}$$

Satz 5.33

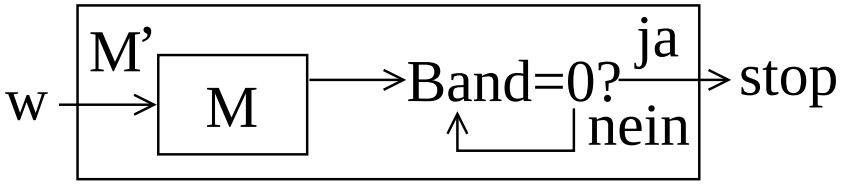
Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.
Dann ist auch folgende Funktion f berechenbar:

$$f(w) := \begin{cases} 0 & \text{falls } \chi_K(w) = 0 \\ \perp & \text{falls } \chi_K(w) = 1 \end{cases}$$

In det Tat, berechnet eine TM M die Funktion χ_K so berechnet die folgende TM M' die Funktion f:



Satz 5.33

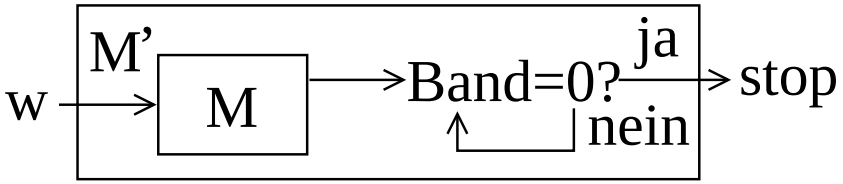
Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.
Dann ist auch folgende Funktion f berechenbar:

$$f(w) := \begin{cases} 0 & \text{falls } \chi_K(w) = 0 \\ \perp & \text{falls } \chi_K(w) = 1 \end{cases}$$

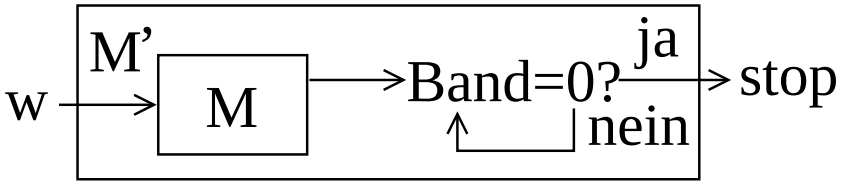
In det Tat, berechnet eine TM M die Funktion χ_K so berechnet die folgende TM M' die Funktion f:



Dann gibt es ein w' mit $M_{w'} = M'$.

Beweis (Forts.):

(Forts.)



$$
\begin{aligned}
f(w') = \perp \quad &\Leftrightarrow \quad \chi_K(w') = 1 && \text{(Def. von } f\text{)} \\
&\Leftrightarrow \quad w' \in K && \text{(Def. von } \chi_K\text{)} \\
&\Leftrightarrow \quad M_{w'}[w']\downarrow && \text{(Def. von } K\text{)} \\
&\Leftrightarrow \quad M'[w']\downarrow && (M_{w'} = M') \\
&\Leftrightarrow \quad f(w') \neq \perp && (M' \text{ berechnet } f)
\end{aligned}
$$

Wir erhalten einen Widerspruch: $f(w') = \perp \Leftrightarrow f(w') \neq \perp$.
Die Annahme, K sei entscheidbar, ist damit falsch.

$\square$

Definition 5.34 ((Allgemeines) Halteproblem)

Gegeben: Wörter $w, x \in \{0,1\}^*$.

Problem: Hält M_w bei Eingabe x ?

Als Menge:

$$H := \{w\#x \mid M_w[x]\downarrow\}$$

Satz 5.35

Das Halteproblem H ist nicht entscheidbar.

Beweis:

Wäre H entscheidbar, dann trivialerweise auch K:

$$\chi_K(w) = \chi_H(w, w)$$

$\square$

Definition 5.36 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist reduzierbar auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*.\ w \in A \Leftrightarrow f(w) \in B$$

Wir schreiben dann $A \leq B$.

Definition 5.36 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist reduzierbar auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*.\ w \in A \Leftrightarrow f(w) \in B$$

Wir schreiben dann $A \leq B$.

Intuition:

- B ist mindestens so schwer zu lösen wie A.
- Ist A unlösbar, dann auch B.
- Ist B lösbar, dann erst recht A.

Lemma 5.37

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Lemma 5.37

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:

Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.
Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \qquad \square$$

Lemma 5.37

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:

Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.
Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \qquad \square$$

Korollar 5.38

Falls $A \leq B$ und A ist unentscheidbar, dann ist auch B unentscheidbar.

Lemma 5.37

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:

Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.
Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \qquad \square$$

Korollar 5.38

Falls $A \leq B$ und A ist unentscheidbar, dann ist auch B unentscheidbar.

Beispiel 5.39

Da $K \leq H$ (mit Reduktion $f(w) := w\#w$) und K unentscheidbar ist, ist auch H unentscheidbar.

Satz 5.40

Das Halteproblem auf leerem Band, H_0, ist unentscheidbar.

$$H_0 := \{w \in \{0,1\}^* \mid M_w[\epsilon]\downarrow\}$$

Satz 5.40

Das Halteproblem auf leerem Band, H_0, ist unentscheidbar.

$$H_0 := \{w \in \{0,1\}^* \mid M_w[\epsilon]\downarrow\}$$

Beweis:

Wir zeigen $K \leq H_0$ mit einer Funktion $f : \{0,1\}^* \to \{0,1\}^*$
$f(w)$ ist die Kodierung folgender TM:

Überschreibe die Eingabe mit w; führe M_w aus.

Dh f berechnet aus w die Kodierung w_1 einer TM, die w schreibt, und gibt die Kodierung von "$w_1; w$" zurück.
Damit ist f total und berechenbar.

Satz 5.40

Das Halteproblem auf leerem Band, H_0, ist unentscheidbar.

$$H_0 := \{w \in \{0,1\}^* \mid M_w[\epsilon]\downarrow\}$$

Beweis:

Wir zeigen $K \leq H_0$ mit einer Funktion $f : \{0,1\}^* \to \{0,1\}^*$
$f(w)$ ist die Kodierung folgender TM:

Überschreibe die Eingabe mit w; führe M_w aus.

Dh f berechnet aus w die Kodierung w_1 einer TM, die w schreibt, und gibt die Kodierung von "$w_1; w$" zurück.
Damit ist f total und berechenbar.
Es gilt:

$$w \in K \;\Leftrightarrow\; M_w[w]\downarrow \;\Leftrightarrow\; M_{f(w)}[\epsilon]\downarrow \;\Leftrightarrow\; f(w) \in H_0$$

$\square$

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
  Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
  Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.
- Kann Variable x_7 bei einer bestimmten Eingabe je den Wert 2^{32} erreichen?

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
  Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.
- Kann Variable x_7 bei einer bestimmten Eingabe je den Wert 2^{32} erreichen?
  Reduktion: Ein Programm P hält gdw

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
  Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.
- Kann Variable x_7 bei einer bestimmten Eingabe je den Wert 2^{32} erreichen?
  Reduktion: Ein Programm P hält gdw während der Ausführung von

  $$P\,;\ x_7 \ := \ 2^{32}$$

  Variable x_7 den Wert 2^{32} erreicht.
  (OE: x_7 kommt in P nicht vor)

5.3 Semi-Entscheidbarkeit

5.3 Semi-Entscheidbarkeit

Definition 5.41

Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt semi-entscheidbar (s-e) gdw

$$\chi'_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ \perp & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Satz 5.42

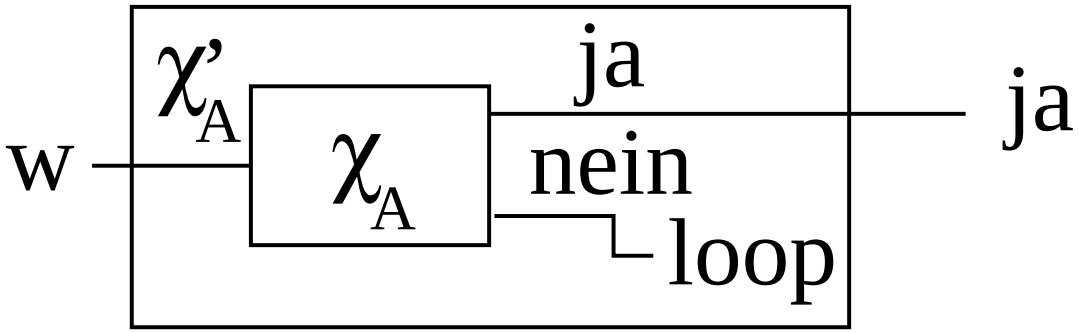
Eine Menge A ist entscheidbar gdw sowohl A als auch $\overline{A}$ s-e sind.

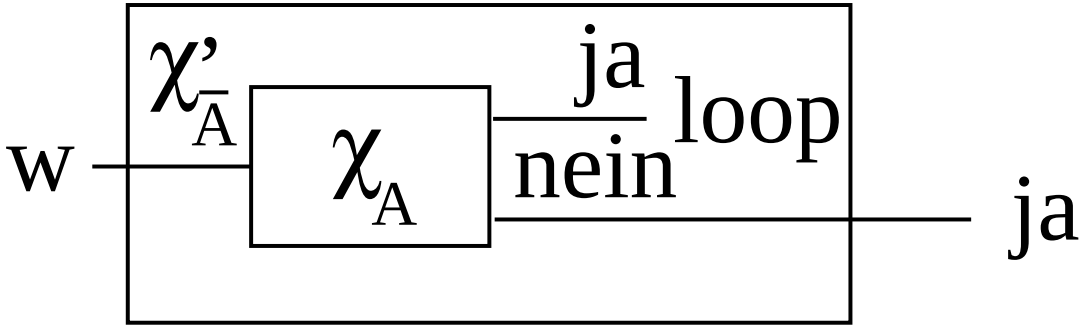
Satz 5.42

Eine Menge A ist entscheidbar gdw sowohl A als auch $\overline{A}$ s-e sind.

Beweis:

„$\Rightarrow$“: Wandle TM für χ_A in TM für χ'_A und $\chi'_{\overline{A}}$ um:





Beweis (Forts.):

„$\Leftarrow$“:

Wandle TM M_1 für χ'_A und TM M_2 für $\chi'_{\overline{A}}$ in TM für χ_A um:

input(x);

for $s := 0, 1, 2, \ldots$ **do**

  if $M_1[x]$ hält in s Schritten **then** output(1); **halt** **fi** ;

  if $M_2[x]$ hält in s Schritten **then** output(0); **halt** **fi**

Beweis (Forts.):

„$\Leftarrow$“:
Wandle TM M_1 für χ'_A und TM M_2 für $\chi'_{\overline{A}}$ in TM für χ_A um:

input(x);
for $s := 0, 1, 2, \ldots$ **do**
  if $M_1[x]$ hält in s Schritten **then** output(1); **halt** **fi** ;
  if $M_2[x]$ hält in s Schritten **then** output(0); **halt** **fi**

Formulierung mit Parallelismus:

input(x);
führe $M_1[x]$ und $M_2[x]$ parallel aus;
hält M_1, gib 1 aus, hält M_2, gib 0 aus. $\square$

Beweis (Forts.):

„$\Leftarrow$“:
Wandle TM M_1 für χ'_A und TM M_2 für $\chi'_{\overline{A}}$ in TM für χ_A um:

input(x);
for $s := 0, 1, 2, \ldots$ **do**
  if $M_1[x]$ hält in s Schritten **then** output(1); **halt** **fi** ;
  if $M_2[x]$ hält in s Schritten **then** output(0); **halt** **fi**

Formulierung mit Parallelismus:

input(x);
führe $M_1[x]$ und $M_2[x]$ parallel aus;
hält M_1, gib 1 aus, hält M_2, gib 0 aus. $\square$

Lemma 5.43

Ist $A \leq B$ und ist B s-e, so ist auch A s-e.

Beweis: Übung

Definition 5.44

Eine Menge A heißt rekursiv aufzählbar (*recursively enumerable*) gdw $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \to A$ gibt, so dass

$$A = \{f(0), f(1), f(2), \ldots\}$$

Definition 5.44

Eine Menge A heißt rekursiv aufzählbar (*recursively enumerable*) gdw $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \to A$ gibt, so dass

$$A = \{f(0), f(1), f(2), \ldots\}$$

Bemerkung:

- Es dürfen Elemente doppelt auftreten ($f(i) = f(j)$ für $i \neq j$)
- Die Reihenfolge ist beliebig.

Definition 5.44

Eine Menge A heißt rekursiv aufzählbar (*recursively enumerable*) gdw $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \to A$ gibt, so dass

$$A = \{f(0), f(1), f(2), \ldots\}$$

Bemerkung:

- Es dürfen Elemente doppelt auftreten ($f(i) = f(j)$ für $i \neq j$)
- Die Reihenfolge ist beliebig.

Warnung: Rekursiv aufzählbar $\neq$ abzählbar!

Definition 5.44

Eine Menge A heißt rekursiv aufzählbar (*recursively enumerable*) gdw $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \to A$ gibt, so dass

$$A = \{f(0), f(1), f(2), \ldots\}$$

Bemerkung:

- Es dürfen Elemente doppelt auftreten ($f(i) = f(j)$ für $i \neq j$)
- Die Reihenfolge ist beliebig.

Warnung: Rekursiv aufzählbar $\neq$ abzählbar!

- Rekursiv aufzählbar $\implies$ abzählbar

Definition 5.44

Eine Menge A heißt rekursiv aufzählbar (*recursively enumerable*) gdw $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \to A$ gibt, so dass

$$A = \{f(0), f(1), f(2), \ldots\}$$

Bemerkung:

- Es dürfen Elemente doppelt auftreten ($f(i) = f(j)$ für $i \neq j$)
- Die Reihenfolge ist beliebig.

Warnung: Rekursiv aufzählbar $\neq$ abzählbar!

- Rekursiv aufzählbar $\implies$ abzählbar
- Aber nicht umgekehrt:
  jede Sprache ist abzählbar aber nicht jede Sprache ist rekursiv aufzählbar (s.u.)

Lemma 5.45

Eine Menge A ist rekursiv aufzählbar gdw sie semi-entscheidbar ist.

Lemma 5.45

Eine Menge A ist rekursiv aufzählbar gdw sie semi-entscheidbar ist.

Beweis:

Der Fall $A = \emptyset$ ist trivial. Sei $A \neq \emptyset$.

Lemma 5.45

Eine Menge A ist rekursiv aufzählbar gdw sie semi-entscheidbar ist.

Beweis:

Der Fall $A = \emptyset$ ist trivial. Sei $A \neq \emptyset$.

„$\Rightarrow$“: Sei A rekursiv aufzählbar mit f. Dann ist A semi-entscheidbar:

input(x);
for $i := 0, 1, 2, \ldots$ **do**
  if $f(i) = x$ **then** output(1); **halt** **fi**

Lemma 5.45

Eine Menge A ist rekursiv aufzählbar gdw sie semi-entscheidbar ist.

Beweis:

Der Fall $A = \emptyset$ ist trivial. Sei $A \neq \emptyset$.

„$\Rightarrow$“: Sei A rekursiv aufzählbar mit f. Dann ist A semi-entscheidbar:

input(x);
for $i := 0, 1, 2, \ldots$ **do**
  if $f(i) = x$ **then** output(1); **halt** **fi**

„$\Leftarrow$“: O.B.d.A. nehmen wir $A \subseteq \mathbb{N}$ an.
Sei A semi-entscheidbar durch (zB) GOTO-Programm P.
Problem: $P[i]$ muss nicht halten und darf daher nur „zeitbeschränkt“ ausgeführt werden. Gesucht: Paare (i, j) so dass $P[i]$ nach j Schritten hält.

Beweis (Forts.):

Idee: Wir benutzen eine geeignete bijektion $c\colon \mathbb{N} \times \mathbb{N} \leftrightarrow \mathbb{N}$.
Seien $p_1 : \mathbb{N} \to \mathbb{N}$ und $p_2 : \mathbb{N} \to \mathbb{N}$ mit

$$p_1(c(n_1, n_2)) = n_1 \qquad \text{und} \qquad p_2(c(n_1, n_2)) = n_2$$

(Umkehrung von c).

Sei $d \in A$ beliebig.
Folgender Algorithmus berechnet eine Aufzählung von A:

input(n);
if $P[p_1(n)]$ hält nach $p_2(n)$ Schritten **then** output($p_1(n)$)
else output(d) **fi**

Beweis (Forts.):

Idee: Wir benutzen eine geeignete bijektion $c\colon \mathbb{N} \times \mathbb{N} \leftrightarrow \mathbb{N}$.
Seien $p_1 : \mathbb{N} \to \mathbb{N}$ und $p_2 : \mathbb{N} \to \mathbb{N}$ mit

$$p_1(c(n_1, n_2)) = n_1 \qquad \text{und} \qquad p_2(c(n_1, n_2)) = n_2$$

(Umkehrung von c).

Sei $d \in A$ beliebig.
Folgender Algorithmus berechnet eine Aufzählung von A:

input(n);
if $P[p_1(n)]$ hält nach $p_2(n)$ Schritten **then** output($p_1(n)$)
else output(d) **fi**

Korrektheit: Der Algorithmus hält immer und liefert immer ein Element aus A.

Beweis (Forts.):

Idee: Wir benutzen eine geeignete bijektion $c\colon \mathbb{N} \times \mathbb{N} \leftrightarrow \mathbb{N}$.
Seien $p_1 : \mathbb{N} \to \mathbb{N}$ und $p_2 : \mathbb{N} \to \mathbb{N}$ mit

$$p_1(c(n_1, n_2)) = n_1 \qquad \text{und} \qquad p_2(c(n_1, n_2)) = n_2$$

(Umkehrung von c).

Sei $d \in A$ beliebig.
Folgender Algorithmus berechnet eine Aufzählung von A:

input(n);
if $P[p_1(n)]$ hält nach $p_2(n)$ Schritten **then** output($p_1(n)$)
else output(d) **fi**

Korrektheit: Der Algorithmus hält immer und liefert immer ein Element aus A.

Vollständigktheit: Sei $a \in A \subseteq \mathbb{N}$.
Dann hält $P[a]$ nach einer endlichen Zahl k von Schritten. Dann liefert die Eingabe $n = c(a, k)$ die Ausgabe a. $\square$

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar
- χ'_A ist berechenbar

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar
- χ'_A ist berechenbar
- $A = L(M)$ für eine TM M

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar
- χ'_A ist berechenbar
- $A = L(M)$ für eine TM M
- A ist Definitionsbereich einer berechenbaren Funktion

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar
- χ'_A ist berechenbar
- $A = L(M)$ für eine TM M
- A ist Definitionsbereich einer berechenbaren Funktion
- A ist Wertebereich einer berechenbaren Funktion

Satz 5.46

Die Menge $K = \{w \mid M_w[w]\downarrow\}$ *ist semi-entscheidbar.*

Satz 5.46

Die Menge $K = \{w \mid M_w[w]\downarrow\}$ ist semi-entscheidbar.

Beweis:

Die Funktion χ'_K ist wie folgt Turing-berechenbar:

Bei Eingabe w simuliere die Ausführung von $M_w[w]$;
gib 1 aus. $\square$

- Hier haben wir benutzt, dass man einen Interpreter/Simulator für Turingmaschinen als Turingmaschine programmieren kann.
- Ein solcher Interpreter wird oft eine Universelle Turingmaschine (U) genannt.

Satz 5.46

Die Menge $K = \{w \mid M_w[w]\downarrow\}$ ist semi-entscheidbar.

Beweis:

Die Funktion χ'_K ist wie folgt Turing-berechenbar:

Bei Eingabe w simuliere die Ausführung von $M_w[w]$;
gib 1 aus. $\square$

- Hier haben wir benutzt, dass man einen Interpreter/Simulator für Turingmaschinen als Turingmaschine programmieren kann.
- Ein solcher Interpreter wird oft eine Universelle Turingmaschine (U) genannt.

Korollar 5.47

$\overline{K}$ ist nicht semi-entscheidbar.

Satz 5.46

Die Menge $K = \{w \mid M_w[w]\downarrow\}$ ist semi-entscheidbar.

Beweis:

Die Funktion χ'_K ist wie folgt Turing-berechenbar:

Bei Eingabe w simuliere die Ausführung von $M_w[w]$;
gib 1 aus. $\square$

- Hier haben wir benutzt, dass man einen Interpreter/Simulator für Turingmaschinen als Turingmaschine programmieren kann.
- Ein solcher Interpreter wird oft eine Universelle Turingmaschine (U) genannt.

Korollar 5.47

$\overline{K}$ ist nicht semi-entscheidbar.

Semi-Entscheidbarkeit ist nicht abgeschlossen unter Komplement.

5.4 Die Sätze von Rice und Shapiro

Die von der TM M_w berechnete Funktion bezeichnen wir mit φ_w.
Wir betrachten implizit nur einstellige Funktionen.

5.4 Die Sätze von Rice und Shapiro

Die von der TM M_w berechnete Funktion bezeichnen wir mit φ_w.
Wir betrachten implizit nur einstellige Funktionen.

Satz 5.48 (Rice)

Sei F eine Menge berechenbarer Funktionen.
Es gelte weder $F = \emptyset$ noch $F =$ alle ber. Funkt. („F nicht trivial“)
Dann ist unentscheidbar, ob die von einer gegebenen TM M_w berechnete Funktion Element F ist, dh ob $\varphi_w \in F$.

5.4 Die Sätze von Rice und Shapiro

Die von der TM M_w berechnete Funktion bezeichnen wir mit φ_w.
Wir betrachten implizit nur einstellige Funktionen.

Satz 5.48 (Rice)

Sei F eine Menge berechenbarer Funktionen.
Es gelte weder $F = \emptyset$ noch $F =$ alle ber. Funkt. („F nicht trivial“)
Dann ist unentscheidbar, ob die von einer gegebenen TM M_w berechnete Funktion Element F ist, dh ob $\varphi_w \in F$.

Alle nicht-triviale semantische Eigenschaften von Programmen sind unentscheidbar.

5.4 Die Sätze von Rice und Shapiro

Die von der TM M_w berechnete Funktion bezeichnen wir mit φ_w.
Wir betrachten implizit nur einstellige Funktionen.

Satz 5.48 (Rice)

Sei F eine Menge berechenbarer Funktionen.
Es gelte weder $F = \emptyset$ noch $F =$ alle ber. Funkt. („F nicht trivial“)
Dann ist unentscheidbar, ob die von einer gegebenen TM M_w berechnete Funktion Element F ist, dh ob $\varphi_w \in F$.

Alle nicht-triviale semantische Eigenschaften von Programmen sind unentscheidbar.

Beispiel 5.49

Es ist unentscheidbar, ob ein Programm

- für mindestens eine Eingabe hält.
  $(F = \{\varphi_w \mid \exists x.\ M_w[x]\downarrow\})$
- für alle Eingaben hält. $(F = \{\varphi_w \mid \forall x.\ M_w[x]\downarrow\})$
- bei Eingabe 42 Ausgabe 42 produziert.

Warnung

Es ist entscheidbar, ob ein Programm

- länger als 5 Zeilen ist.
- eine Zuweisung an die Variable x_{17} enhält.

Im Satz von Rice geht es um die von einem Programm berechnete Funktion (Semantik),
nicht um den Programmtext (Syntax).

Beweis:

Wir zeigen $C_F := \{w \in \{0,1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Beweis:

Wir zeigen $C_F := \{w \in \{0,1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Fall 1: $\Omega := (x \mapsto \bot) \notin F$.

Wähle $h \in F \neq \emptyset$ beliebig; sei u Kodierung einer TM mit $\varphi_u = h$.

Beweis:

Wir zeigen $C_F := \{w \in \{0,1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Fall 1: $\Omega := (x \mapsto \bot) \notin F$.

Wähle $h \in F \neq \emptyset$ beliebig; sei u Kodierung einer TM mit $\varphi_u = h$.
Reduziere K auf C_F ($K \leq C_F$) mit $f : \{0,1\}^* \to \{0,1\}^*$ und $f(w)$ die Kodierung folgender TM:

Speichere die Eingabe x auf einem getrennten Band; schreibe $w\#w$ auf die Eingabe; führe die universelle TM U auf $w\#w$ aus; führe M_u auf x aus.

Beweis:

Wir zeigen $C_F := \{w \in \{0,1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Fall 1: $\Omega := (x \mapsto \bot) \notin F$.

Wähle $h \in F \neq \emptyset$ beliebig; sei u Kodierung einer TM mit $\varphi_u = h$.
Reduziere K auf C_F ($K \leq C_F$) mit $f : \{0,1\}^* \to \{0,1\}^*$ und $f(w)$ die Kodierung folgender TM:

Speichere die Eingabe x auf einem getrennten Band; schreibe $w\#w$ auf die Eingabe; führe die universelle TM U auf $w\#w$ aus; führe M_u auf x aus.

Es gilt $\varphi_{f(w)} = \begin{cases} h & \text{falls } M_w[w]\downarrow \\ \Omega & \text{sonst} \end{cases}$ und damit

$$w \in K \;\Leftrightarrow\; M_w[w]\downarrow \;\overset{(*)}{\Leftrightarrow}\; \varphi_{f(w)} \in F \;\Leftrightarrow\; f(w) \in C_F$$

Beweis:

Wir zeigen $C_F := \{w \in \{0,1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Fall 1: $\Omega := (x \mapsto \bot) \notin F$.

Wähle $h \in F \neq \emptyset$ beliebig; sei u Kodierung einer TM mit $\varphi_u = h$.
Reduziere K auf C_F ($K \leq C_F$) mit $f : \{0,1\}^* \to \{0,1\}^*$ und $f(w)$ die Kodierung folgender TM:

Speichere die Eingabe x auf einem getrennten Band; schreibe $w\#w$ auf die Eingabe; führe die universelle TM U auf $w\#w$ aus; führe M_u auf x aus.

Es gilt $\varphi_{f(w)} = \begin{cases} h & \text{falls } M_w[w]\downarrow \\ \Omega & \text{sonst} \end{cases}$ und damit

$$w \in K \;\Leftrightarrow\; M_w[w]\downarrow \;\overset{(*)}{\Leftrightarrow}\; \varphi_{f(w)} \in F \;\Leftrightarrow\; f(w) \in C_F$$

$$(*) : \begin{cases} M_w[w]\downarrow \Rightarrow \varphi_{f(w)} = h \in F \\ \varphi_{f(w)} \in F \Rightarrow \varphi_{f(w)} = h \Rightarrow M_w[w]\downarrow \end{cases}$$

Beweis (Forts.):

Fall 2: $\Omega \in F$.

Wähle berechenbares $h \notin F$.

Zeige analog, dass $\overline{K} \leq C_F$. $\square$

Satz 5.50 (Rice-Shapiro)

Sei F eine Menge berechenbarer Funktionen.
Ist $C_F := \{w \mid \varphi_w \in F\}$ semi-entscheidbar,
so gilt für alle berechenbaren f:
$f \in F \Leftrightarrow$ es gibt eine endliche Teilfunktion $g \subseteq f$ mit $g \in F$.

Satz 5.50 (Rice-Shapiro)

Sei F eine Menge berechenbarer Funktionen.
Ist $C_F := \{w \mid \varphi_w \in F\}$ semi-entscheidbar,
so gilt für alle berechenbaren f:
$f \in F \Leftrightarrow$ es gibt eine endliche Teilfunktion $g \subseteq f$ mit $g \in F$.

Beweis:

„$\Rightarrow$“ mit Widerspruch.
Sei $f \in F$, so dass für alle endlichen $g \subseteq f$ gilt $g \notin F$.
Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist. `

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ ist die Kodierung folgender TM:

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t simuliere t Schritte von $M_w[w]$.
Hält diese Berechnung in $\leq t$ Schritten, gehe in eine endlos Schleife, sonst berechne $f(t)$.

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t simuliere t Schritte von $M_w[w]$.
Hält diese Berechnung in $\leq t$ Schritten, gehe in eine endlos Schleife, sonst berechne $f(t)$.

Wir zeigen

$$w \in \overline{K} \;\Leftrightarrow\; h(w) \in C_F$$

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t simuliere t Schritte von $M_w[w]$.
Hält diese Berechnung in $\leq t$ Schritten, gehe in eine endlos Schleife, sonst berechne $f(t)$.

Wir zeigen

$$w \in \overline{K} \;\Leftrightarrow\; h(w) \in C_F$$

- $w \in \overline{K} \implies \neg M_w[w]\downarrow \implies \varphi_{h(w)} = f \in F \implies h(w) \in C_F$

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t simuliere t Schritte von $M_w[w]$.
Hält diese Berechnung in $\leq t$ Schritten, gehe in eine endlos Schleife, sonst berechne $f(t)$.

Wir zeigen

$$w \in \overline{K} \;\Leftrightarrow\; h(w) \in C_F$$

- $w \in \overline{K} \implies \neg M_w[w]\downarrow \implies \varphi_{h(w)} = f \in F \implies h(w) \in C_F$
- Falls $w \notin \overline{K}$ dann hält $M_w[w]$ nach eine Zahl t von Schritten.
  Damit gilt: $\varphi_{h(w)}$ ist f eingeschränkt auf $\{0, \ldots, t-1\}$.
  Nach Annahme folgt $\varphi_{h(w)} \notin F$, dh $h(w) \notin C_F$.

Beweis (Forts.):

„$\Leftarrow$“ mit Widerspruch.

Beweis (Forts.):

„$\Leftarrow$“ mit Widerspruch.
Sei f berechenbar, sei $g \subseteq f$ endlich mit $g \in F$, aber sei $f \notin F$.
Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist. `
Reduktion $\overline{K} \leq C_F$ mit $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ ist die Kodierung folgender TM:

Beweis (Forts.):

„$\Leftarrow$“ mit Widerspruch.
Sei f berechenbar, sei $g \subseteq f$ endlich mit $g \in F$, aber sei $f \notin F$.
Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist. `
Reduktion $\overline{K} \leq C_F$ mit $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t, teste ob t im *endlichen* Def. ber. von g ist.
Wenn ja, berechne $f(t)$,
sonst simuliere $M_w[w]$ und berechne dann $f(t)$.

Beweis (Forts.):

„$\Leftarrow$“ mit Widerspruch.
Sei f berechenbar, sei $g \subseteq f$ endlich mit $g \in F$, aber sei $f \notin F$.
Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist.
Reduktion $\overline{K} \leq C_F$ mit $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ ist die Kodierung folgender TM:

> Bei Eingabe t, teste ob t im *endlichen* Def. ber. von g ist.
> Wenn ja, berechne $f(t)$,
> sonst simuliere $M_w[w]$ und berechne dann $f(t)$.

Wir zeigen

$$w \in \overline{K} \;\Leftrightarrow\; h(w) \in C_F$$

Beweis (Forts.):

„$\Leftarrow$“ mit Widerspruch.
Sei f berechenbar, sei $g \subseteq f$ endlich mit $g \in F$, aber sei $f \notin F$.
Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist.
Reduktion $\overline{K} \leq C_F$ mit $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ ist die Kodierung folgender TM:

> Bei Eingabe t, teste ob t im *endlichen* Def. ber. von g ist.
> Wenn ja, berechne $f(t)$,
> sonst simuliere $M_w[w]$ und berechne dann $f(t)$.

Wir zeigen

$$w \in \overline{K} \;\Leftrightarrow\; h(w) \in C_F$$

- $w \in \overline{K} \implies \neg M_w[w]\downarrow \implies \varphi_{h(w)} = g \in F \implies h(w) \in C_F$
- $w \notin \overline{K} \implies M_w[w]\downarrow \implies \varphi_{h(w)} = f \notin F \implies h(w) \notin C_F$

$\square$

Rice-Shapiro (in Kurzform): $C_F := \{w \mid \varphi_w \in F\}$ s-e $\Longrightarrow$

$f \in F \Leftrightarrow$ es gibt endliche Funkt. $g \subseteq f$ mit $g \in F$.

Rice-Shapiro (in Kurzform): $C_F := \{w \mid \varphi_w \in F\}$ s-e $\Longrightarrow$

$$f \in F \Leftrightarrow \text{es gibt endliche Funkt. } g \subseteq f \text{ mit } g \in F.$$

Ein Programm heißt terminierend gdw es für alle Eingaben hält.

Korollar 5.51

- *Die Menge der terminierenden Programme ist nicht semi-entscheidbar.*
- *Die Menge der nicht-terminierenden Programme ist nicht semi-entscheidbar.*

Rice-Shapiro (in Kurzform): $C_F := \{w \mid \varphi_w \in F\}$ s-e $\Longrightarrow$

$$f \in F \Leftrightarrow \text{es gibt endliche Funkt. } g \subseteq f \text{ mit } g \in F.$$

Ein Programm heißt terminierend gdw es für alle Eingaben hält.

Korollar 5.51

- *Die Menge der terminierenden Programme ist nicht semi-entscheidbar.*
- *Die Menge der nicht-terminierenden Programme ist nicht semi-entscheidbar.*

Beweis:

- $F :=$ Menge aller berechenbaren totalen Funktionen.
  Sei $f \in F$. Jede endliche $g \subseteq f$ ist echt partiell, dh $g \notin F$.
  Also kann C_F nicht semi-entscheidbar sein.

Rice-Shapiro (in Kurzform): $C_F := \{w \mid \varphi_w \in F\}$ s-e $\Longrightarrow$

$$f \in F \Leftrightarrow \text{es gibt endliche Funkt. } g \subseteq f \text{ mit } g \in F.$$

Ein Programm heißt terminierend gdw es für alle Eingaben hält.

Korollar 5.51

- *Die Menge der terminierenden Programme ist nicht semi-entscheidbar.*
- *Die Menge der nicht-terminierenden Programme ist nicht semi-entscheidbar.*

Beweis:

- $F :=$ Menge aller berechenbaren totalen Funktionen.
  Sei $f \in F$. Jede endliche $g \subseteq f$ ist echt partiell, dh $g \notin F$.
  Also kann C_F nicht semi-entscheidbar sein.
- $F :=$ Menge aller berechenbaren nicht-totalen Funktionen.
  Sei f total und berechenbar. Damit $f \notin F$.
  Aber jede endliche $g \subseteq f$ ist in F.
  Also kann C_F nicht semi-entscheidbar sein. $\square$

Grenzen automatischer Terminationsanalyse von Programmen

Grenzen automatischer Terminationsanalyse von Programmen

- Termination ist unentscheidbar (Rice):
  Klare Ja/Nein Antwort unmöglich.

Grenzen automatischer Terminationsanalyse von Programmen

- Termination ist unentscheidbar (Rice):
  Klare Ja/Nein Antwort unmöglich.
- Termination ist nicht semi-entscheidbar (Rice-Shapiro):
  Es gibt kein Zertifizierungs-Programm,
  das alle terminierenden Programme erkennt.

Grenzen automatischer Terminationsanalyse von Programmen

- Termination ist unentscheidbar (Rice):
  Klare Ja/Nein Antwort unmöglich.
- Termination ist nicht semi-entscheidbar (Rice-Shapiro):
  Es gibt kein Zertifizierungs-Programm,
  das alle terminierenden Programme erkennt.
- Nicht-Termination ist nicht semi-entscheidbar (Rice-Shapiro):
  Es gibt keinen perfekten *Bug Finder*,
  der alle nicht-terminierenden Programme erkennt.

Grenzen automatischer Terminationsanalyse von Programmen

- Termination ist unentscheidbar (Rice):
  Klare Ja/Nein Antwort unmöglich.
- Termination ist nicht semi-entscheidbar (Rice-Shapiro):
  Es gibt kein Zertifizierungs-Programm,
  das alle terminierenden Programme erkennt.
- Nicht-Termination ist nicht semi-entscheidbar (Rice-Shapiro):
  Es gibt keinen perfekten *Bug Finder*,
  der alle nicht-terminierenden Programme erkennt.

Aber es gibt mächtige heuristische Verfahren, die für
relativ viele Programme aus der Praxis (Gerätetreiber)

- Termination beweisen können, oder
- Gegenbeispiele finden können.

5.5 Das Postsche Korrespondenzproblem

Gegeben beliebig viele Kopien der 3 „Spielkarten“

001	10	0
00	11	010

5.5 Das Postsche Korrespondenzproblem

Gegeben beliebig viele Kopien der 3 „Spielkarten“

001	10	0
00	11	010

gibt es dann eine Folge dieser Karten

...	...	...
...		...

so dass oben und unten das gleiche Wort steht?

5.5 Das Postsche Korrespondenzproblem

Gegeben beliebig viele Kopien der 3 „Spielkarten“

001	10	0
00	11	010

gibt es dann eine Folge dieser Karten

...	...	...
...		...

so dass oben und unten das gleiche Wort steht?

001	10	001	0
00	11	00	010

Kurz: 1,2,1,3.

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \ldots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, $n > 0$, mit $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$?

Dann nennen wir $i_1, \ldots, i_n$ eine Lösung des Problems $(x_1, y_1), \ldots, (x_k, y_k)$.

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \ldots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, $n > 0$, mit $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$?

Dann nennen wir $i_1, \ldots, i_n$ eine Lösung des Problems $(x_1, y_1), \ldots, (x_k, y_k)$.

Beispiel 5.53

- Hat $(1, 111)$, $(10111, 10)$, $(10, 0)$ eine Lösung? 2,1,1,3

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \ldots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, $n > 0$, mit $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$?

Dann nennen wir $i_1, \ldots, i_n$ eine Lösung des Problems $(x_1, y_1), \ldots, (x_k, y_k)$.

Beispiel 5.53

- Hat $(1, 111), (10111, 10), (10, 0)$ eine Lösung? 2,1,1,3
- Hat $(b, ca), (a, ab), (ca, a), (abc, c)$ eine Lösung? 2,1,3,2,4

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \ldots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, $n > 0$, mit $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$?

Dann nennen wir $i_1, \ldots, i_n$ eine Lösung des Problems $(x_1, y_1), \ldots, (x_k, y_k)$.

Beispiel 5.53

- Hat $(1, 111)$, $(10111, 10)$, $(10, 0)$ eine Lösung? 2,1,1,3
- Hat $(b, ca), (a, ab), (ca, a), (abc, c)$ eine Lösung? 2,1,3,2,4
- Hat $(101, 01)$, $(101, 010)$, $(010, 10)$ eine Lösung? Nein!

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \ldots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, $n > 0$, mit $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$?

Dann nennen wir $i_1, \ldots, i_n$ eine Lösung des Problems $(x_1, y_1), \ldots, (x_k, y_k)$.

Beispiel 5.53

- Hat $(1, 111)$, $(10111, 10)$, $(10, 0)$ eine Lösung? 2,1,1,3
- Hat $(b, ca), (a, ab), (ca, a), (abc, c)$ eine Lösung? 2,1,3,2,4
- Hat $(101, 01)$, $(101, 010)$, $(010, 10)$ eine Lösung? Nein!
- Hat $(10, 101)$, $(011, 11)$, $(101, 011)$ eine Lösung? [HMU]

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \ldots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, $n > 0$, mit $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$?

Dann nennen wir $i_1, \ldots, i_n$ eine Lösung des Problems $(x_1, y_1), \ldots, (x_k, y_k)$.

Beispiel 5.53

- Hat $(1, 111)$, $(10111, 10)$, $(10, 0)$ eine Lösung? 2,1,1,3
- Hat $(b, ca), (a, ab), (ca, a), (abc, c)$ eine Lösung? 2,1,3,2,4
- Hat $(101, 01)$, $(101, 010)$, $(010, 10)$ eine Lösung? Nein!
- Hat $(10, 101)$, $(011, 11)$, $(101, 011)$ eine Lösung? [HMU]
- Hat $(1000, 10)$, $(1, 0011)$, $(0, 111)$, $(11, 0)$ eine Lösung? Ja, mit Länge 495.

Emil Post.
A Variant of a Recursively Unsolvable Problem.
Bulletin American Mathematical Society, 1946.

Emil Leon Post, 1897 (Polen) – 1954 (NY).

Lemma 5.54

Das PCP ist semi-entscheidbar.

Lemma 5.54

Das PCP ist semi-entscheidbar.

Beweis:

Zähle die möglichen Lösungen der Länge nach auf, und probiere jeweils, ob es eine wirkliche Lösung ist. □

Lemma 5.54

Das PCP ist semi-entscheidbar.

Beweis:

Zähle die möglichen Lösungen der Länge nach auf, und probiere jeweils, ob es eine wirkliche Lösung ist. □

Wir zeigen nun:

$$H \leq MPCP \leq PCP$$

wobei

Definition 5.55 (Modifiziertes PCP, MPCP)

Gegeben: wie beim PCP

Problem: Gibt es eine Lösung $i_1, \ldots, i_n$ mit $i_1 = 1$?

Satz 5.56

$$MPCP \leq PCP$$

Beweis:

Für $w = a_1 \ldots a_n$:

$$\begin{aligned}
\overline{w} &:= \#a_1\#a_2\#\ldots\#a_n\# \\
\overleftarrow{w} &:= a_1\#a_2\#\ldots\#a_n\# \\
\overrightarrow{w} &:= \#a_1\#a_2\#\ldots\#a_n
\end{aligned}$$

$$f((x_1, y_1), \ldots, (x_k, y_k)) := ((\overline{x_1}, \overrightarrow{y_1}), (\overleftarrow{x_1}, \overrightarrow{y_1}), \ldots, (\overleftarrow{x_k}, \overrightarrow{y_k}), (\$, \#\$))$$

Satz 5.57

$$H \leq MPCP$$

Beweis:

- $(\#, \#q_0u\#)$
- (a, a) für alle $a \in \Gamma \cup \{\#\}$
- $(qa, q'a')$ falls $\delta(q, a) = (q', a', N)$
  $(qa, a'q')$ falls $\delta(q, a) = (q', a', R)$
  $(bqa, q'ba')$ falls $\delta(q, a) = (q', a', L)$, für alle $b \in \Gamma$
- $(\#, \Box\#)$, $(\#, \#\Box)$
- (aq, q), (qa, q) für alle $q \in F, a \in \Gamma$
- $(q\#\#, \#)$ für alle $q \in F$

Aus $H \leq PCP$ folgt direkt

Korollar 5.58

Das PCP ist unentscheidbar.

Aus $H \leq PCP$ folgt direkt

Korollar 5.58
Das PCP ist unentscheidbar.

Korollar 5.59
Das PCP ist auch für $\Sigma = \{0, 1\}$ unentscheidbar

Beweis:
Wir nennen dies das 01-PCP und zeigen PCP $\leq$ 01-PCP.
Sei $\Sigma = \{a_1, \ldots, a_m\}$ das Alphabet des gegebenen PCPs.

Aus $H \leq PCP$ folgt direkt

Korollar 5.58

Das PCP ist unentscheidbar.

Korollar 5.59

Das PCP ist auch für $\Sigma = \{0, 1\}$ unentscheidbar

Beweis:

Wir nennen dies das 01-PCP und zeigen PCP $\leq$ 01-PCP.
Sei $\Sigma = \{a_1, \ldots, a_m\}$ das Alphabet des gegebenen PCPs.

Abbildung auf ein 01-PCP:
$$\begin{aligned} \widehat{a_j} &:= 01^j \\ \widehat{a_{j_1} \ldots a_{j_n}} &:= \widehat{a_{j_1}} \ldots \widehat{a_{j_n}} \end{aligned}$$

Aus $H \leq PCP$ folgt direkt

Korollar 5.58

Das PCP ist unentscheidbar.

Korollar 5.59

Das PCP ist auch für $\Sigma = \{0,1\}$ unentscheidbar

Beweis:

Wir nennen dies das 01-PCP und zeigen PCP $\leq$ 01-PCP.
Sei $\Sigma = \{a_1, \ldots, a_m\}$ das Alphabet des gegebenen PCPs.

Abbildung auf ein 01-PCP:
$$\begin{array}{rcl} \widehat{a_j} & := & 01^j \\ \widehat{a_{j_1} \ldots a_{j_n}} & := & \widehat{a_{j_1}} \ldots \widehat{a_{j_n}} \end{array}$$

Dann hat $(x_1, y_1), \ldots, (x_k, y_k)$ eine Lösung gdw
$(\widehat{x_1}, \widehat{y_1}), \ldots, (\widehat{x_k}, \widehat{y_k})$ eine Lösung hat.

„$\Rightarrow$“ klar, „$\Leftarrow$“ folgt da $\hat{\cdot} : \Sigma^* \to \{0,1\}^*$ injektive ist:

$$\begin{array}{lcl} \widehat{x_{i_1}} \ldots \widehat{x_{i_n}} = \widehat{y_{i_1}} \ldots \widehat{y_{i_n}} & \Longrightarrow & \\ \widehat{x_{i_1} \ldots x_{i_n}} = \widehat{y_{i_1} \ldots y_{i_n}} & \Longrightarrow & x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n} \end{array}$$

$\square$

Bemerkungen

- Das PCP ist entscheidbar falls $|\Sigma| = 1$
- Das PCP ist entscheidbar falls $k \leq 2$.
- Das PCP ist unentscheidbar falls $k \geq 7$.
- Für $k = 3, \ldots, 6$ ist noch offen, ob das PCP unentscheidbar ist.

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
  $L(A) = \emptyset$, $L(A) = L(B)$, ...

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
  $L(A) = \emptyset$, $L(A) = L(B)$, ...
- Für TMs ist fast nichts entscheidbar:
  $L(M) = \emptyset$, $L(M_1) = L(M_2)$, ...

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
  $L(A) = \emptyset$, $L(A) = L(B)$, ...
- Für TMs ist fast nichts entscheidbar:
  $L(M) = \emptyset$, $L(M_1) = L(M_2)$, ...
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$, $w \in L(G)$),
  und manches unentscheidbar.

Satz 5.60

Folgendes Problem ist unentscheidbar: Gegeben zwei CGF G_1, G_2, gilt $L(G_1) \cap L(G_2) = \emptyset$?

Beweis:

Reduktion von PCP auf $\{G_1, G_2 \in \text{CFG} \mid L(G_1) \cap L(G_2) = \emptyset\}$.

Beweisidee: Instanz von PCP wird abgebildet auf (G_1, G_2) die jeweils die Wörter folgender Gestalt erzeugen:

Spieg(Kartenseq1) Oben1 *Spieg*(Unten2) Kartenseq2

Spieg(Kartenseq) Wort *Spieg*(Wort) Kartenseq

Der Schnitt $L(G_1 \cap G_2)$ enthält somit alle Wörter der Gestalt

Spieg(Kartenseq1) Oben1 *Spieg*(Unten2) Kartenseq2

mit Kartenseq1=Kartenseq2 und Oben1=Unten2.

Diese Wörter sind die Lösungen der Instanz von PCP.

Beweis (Forts.):

Instanz $(x_1, y_1), \ldots, (x_k, y_k)$ von PCP wird abgebildet auf (G_1, G_2) mit:

$$
\begin{array}{llll}
G_1: & S & \to & A\,\$\,B \\
 & A & \to & a_1 A x_1 \mid \cdots \mid a_k A x_k \\
 & A & \to & a_1 x_1 \mid \cdots \mid a_k x_k \\
 & B & \to & y_1^R B a_1 \mid \cdots \mid y_k^R B a_k \\
 & B & \to & y_1^R a_1 \mid \cdots \mid y_k^R a_k
\end{array}
$$

$$
\begin{array}{llll}
G_2: & S & \to & a_1 S a_1 \mid \cdots \mid a_k S a_k \mid T \\
 & T & \to & 0T0 \mid 1T1 \mid \$
\end{array}
$$

G_1 erzeugt die Sprache

$$\{\ a_{i_n} \cdots a_{i_1} x_{i_1} \cdots x_{i_n}\ \$\ y_{j_m}^R \cdots y_{j_1}^R a_{j_1} \cdots a_{j_m} \mid i_1, \ldots j_m \in [1..k]\ \}$$

G_2 erzeugt die Sprache

$$\{\ a_{i_n} \cdots a_{i_1} b_1\ \cdots\ b_m\ \$\ b_m\ \cdots\ b_1\ a_{i_1} \cdots a_{i_n}\ \mid i_1, \ldots i_n\ \in [1..k]\ \}$$

$\square$

Satz 5.61

Für CFGs G_1, G_2 sind folgende Probleme unentscheidbar:

1. *Ist* $L(G_1) \cap L(G_2) = \emptyset$*?*
2. *Ist* $|L(G_1) \cap L(G_2)| = \infty$ *?*
3. *Ist* $L(G_1) \cap L(G_2)$ *kontextfrei?*
4. *Ist* $L(G_1) \subseteq L(G_2)$*?*
5. *Ist* $L(G_1) = L(G_2)$*?*

Beweis:

Übung. Varianten der Reduktion von PCP auf "Ist $L(G_1) \cap L(G_2) = \emptyset$?" $\square$

Definition 5.62

Eine CFG ist deterministisch (DCFG) gdw $L(G)$ deterministisch ist.

Fakt 5.63

G_1 und G_2 im Beweis zu Satz 5.61 sind deterministisch.

Korollar 5.64

Die Probleme in Satz 5.61 sind bereits für DCFGs unentscheidbar.

Satz 5.65

Für eine CFG G sind folgende Probleme unentscheidbar:

Satz 5.65

Für eine CFG G sind folgende Probleme unentscheidbar:

1. *Ist G mehrdeutig?*
2. *Ist $L(G)$ regulär?*
3. *Ist $L(G)$ deterministisch (DCFL)?*

Satz 5.65

Für eine CFG G sind folgende Probleme unentscheidbar:

1. *Ist G mehrdeutig?*
2. *Ist $L(G)$ regulär?*
3. *Ist $L(G)$ deterministisch (DCFL)?*

Beweis:

1. Reduktion von PCP. Kartenmenge $\mapsto G_3 :=$ „$G_1 \cup G_2$“.
Menge lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig.
„$\Rightarrow$“: Syntaxbäume von G_1 und G_2 disjunkt
„$\Leftarrow$“: $L(G_1)$ und $L(G_2)$ sind DCFL und damit nicht mehrdeutig

Satz 5.65

Für eine CFG G sind folgende Probleme unentscheidbar:

1. *Ist G mehrdeutig?*
2. *Ist $L(G)$ regulär?*
3. *Ist $L(G)$ deterministisch (DCFL)?*

Beweis:

1. Reduktion von PCP. Kartenmenge $\mapsto G_3 :=$ „$G_1 \cup G_2$“.
Menge lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig.
„$\Rightarrow$“: Syntaxbäume von G_1 und G_2 disjunkt
„$\Leftarrow$“: $L(G_1)$ und $L(G_2)$ sind DCFL und damit nicht mehrdeutig

2/3. Reduktion von PCP. Kartenmenge $\mapsto G_4 :=$ „$\overline{G_1} \cup \overline{G_2}$“.
Menge unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset \Rightarrow L(G_4) = \Sigma^*$
$\Rightarrow L(G_4)$ reg/det.
Menge lösbar $\Rightarrow L(G_1) \cap L(G_2)$ nicht CFL$\Rightarrow \overline{L(G_4)}$ nicht reg/det
$\Rightarrow L(G_4)$ nicht reg/det. $\square$

Satz 5.66

Für eine CFG G und einen RE α ist $L(G) = L(\alpha)$ unentscheidbar.

Satz 5.66

Für eine CFG G und einen RE α ist $L(G) = L(\alpha)$ unentscheidbar.

Beweis:

Im Beweis des vorherigen Satzes hatten wir eine Reduktion $PCP \mapsto G_4$ mit

$$PCP \text{ unlösbar } \Leftrightarrow L(G_4) = \Sigma^*.$$

Sei $\Sigma = \{a_1, \ldots, a_n\}$. Dann reduziert $PCP \mapsto (G_4, (a_1| \ldots |a_n)^*)$ das PCP auf das Problem $L(G) = L(\alpha)$, $\square$

6. Komplexitätstheorie

- Was ist mit beschränkten Mitteln (Zeit&Platz) berechenbar?
- Wieviel Zeit&Platz braucht man,
  um ein bestimmtes Problem/Sprache zu entscheiden?
- **Komplexität eines Problems**, nicht eines Algorithmus!

Polynomielle und exponentielle Komplexität

Taktfrequenz: 1GHz

	Problemgröße n					
Zeit	10	20	30	40	50	60
n	.01 ms	.02 ms	.03 ms	.04 ms	.05 ms	.06 ms
n^2	.1 ms	.4 ms	.9 ms	1.6 ms	2.5 ms	3.6 ms
n^5	100ms	0.003s	0.02s	0.1 s	0.3 s	0.7 s
2^n	1 ms	0.001s	1 s	18 m	13 T	36 J
3^n	59 ms	3 s	2 T	385J	$2 \cdot 10^7$ J	10^{12} J

ms=Mikrosek., s=Sek., m=Minute, T=Tag, J=Jahr

Polynomielle und exponentielle Komplexität

Taktfrequenz: 1GHz

Zeit	Problemgröße n					
	10	20	30	40	50	60
n	.01 ms	.02 ms	.03 ms	.04 ms	.05 ms	.06 ms
n^2	.1 ms	.4 ms	.9 ms	1.6 ms	2.5 ms	3.6 ms
n^5	100ms	0.003s	0.02s	0.1 s	0.3 s	0.7 s
2^n	1 ms	0.001s	1 s	18 m	13 T	36 J
3^n	59 ms	3 s	2 T	385J	$2 \cdot 10^7$ J	10^{12} J

ms=Mikrosek., s=Sek., m=Minute, T=Tag, J=Jahr

Problemgröße lösbar in fester Zeit: Speedup

Komplexität	n	n^2	n^5	2^n	3^n
1 MHz Prozessor	N_1	N_2	N_3	N_4	N_5
1 GHz Prozessor	1000 N_1	32 N_2	4 N_3	$N_4 + 10$	$N_5 + 6$

Zentrale Frage: Für welche Probleme gibt es/gibt es keine polynomielle Algorithmen?

Komplexitätsklasse P:

P = die von DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible* problems)

- $A \in P$ wird durch Angabe eines Algorithmus bewiesen
  Bsp: CYK beweist

$$\{(G, w) \mid G \in \text{CFG}, w \in L(G)\} \in \text{P}$$

- $A \notin P$ zu zeigen ist viel schwieriger! Für sehr viele Probleme ist es nicht bekannt, ob sie zu P gehören oder nicht.

Viele der wichtigsten Problem der Informatik sind der Gestalt:

Gegeben X, gibt es Y mit $R(X, Y)$?

- SAT: $X =$ Boole'sche Formel, $Y =$ Belegung der Variablen von X, $R(X, Y) :=$ "Y erfüllt X".
- HAMILTONKREIS: $X =$ Graph, $Y =$ Kreis von X, $R(X, Y) :=$ "Y besucht alle Knoten von X genau einmal".
- EULERKREIS: $X =$ Graph, $Y =$ Kreis von X, $R(X, Y) :=$ "Y besucht alle Kanten von X genau einmal".

Die Y sind "Lösungskandidaten". In allen diesen Problemen:

- Prüfen, ob ein Kandidat tatsächlich eine Lösung ist, ist in P.
- Es gibt jedoch exponentiell viele Kandidaten.
- Deshalb hat ein naïver Suchalgorithmus, der alle Kandidaten aufzählt und prüft, exponentielle Laufzeit.

Für kein solches Problem ist bewiesen worden, dass es *nicht* in P liegt.

Die Frage, ob *alle* solche Probleme in P liegen, ist die wichtigste offene Frage der Informatik.

Für kein solches Problem ist bewiesen worden, dass es *nicht* in P liegt.

Die Frage, ob *alle* solche Probleme in P liegen, ist die wichtigste offene Frage der Informatik.

Äquivalente Formulierung

Diese Probleme können "nichtdeterministisch" in polynomieller Zeit gelöst werden: Wähle einen Kandidaten und prüfe, ob er eine Lösung ist.

Für kein solches Problem ist bewiesen worden, dass es *nicht* in P liegt.

Die Frage, ob *alle* solche Probleme in P liegen, ist die wichtigste offene Frage der Informatik.

Äquivalente Formulierung

Diese Probleme können "nichtdeterministisch" in polynomieller Zeit gelöst werden: Wähle einen Kandidaten und prüfe, ob er eine Lösung ist.

Komplexitätsklasse NP:

NP = die von NTM in polynomieller Zeit lösbaren Probleme

Zentrale Frage:

$$\mathrm{P} = \mathrm{NP}\ ?$$

Überblick:

1. Die Komplexitätsklassen P und NP
2. NP-Vollständigkeit
3. SAT: Ein NP-vollständiges Problem
4. Weitere NP-vollständige Probleme

6.1 Die Komplexitätsklasse P

Berechnungsmodell:

DTM = deterministische Mehrband-TM

6.1 Die Komplexitätsklasse P

Berechnungsmodell:

DTM = deterministische Mehrband-TM

Definition 6.1

$\mathit{time}_M(w)$:= Anzahl der Schritte bis die DTM $M[w]$ hält $\in \mathbb{N} \cup \{\infty\}$

6.1 Die Komplexitätsklasse P

Berechnungsmodell:

DTM = deterministische Mehrband-TM

Definition 6.1

$$\mathit{time}_M(w) \quad := \text{Anzahl der Schritte bis die DTM } M[w] \text{ hält} \in \mathbb{N} \cup \{\infty\}$$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.
Klasse der in Zeit $f(n)$ entscheidbaren Sprachen:

$$\mathit{TIME}(f(n)) \quad := \quad \{A \subseteq \Sigma^* \quad | \quad \exists \text{DTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ \mathit{time}_M(w) \leq f(|w|)\}$$

6.1 Die Komplexitätsklasse P

Berechnungsmodell:

DTM = deterministische Mehrband-TM

Definition 6.1

$$\mathit{time}_M(w) \quad := \text{Anzahl der Schritte bis die DTM } M[w] \text{ hält} \in \mathbb{N} \cup \{\infty\}$$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.
Klasse der in Zeit $f(n)$ entscheidbaren Sprachen:

$$\mathit{TIME}(f(n)) \quad := \quad \{A \subseteq \Sigma^* \quad | \quad \exists \text{DTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ \mathit{time}_M(w) \leq f(|w|)\}$$

Merke:

- n ist implizit die Länge der Eingabe
- Die DTM entscheidet die Sprache A in $\leq f(n)$ Schritten

Wir betrachten nur Polynome mit Koeffizienten $a_k, \ldots a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \cdots + a_1 n + a_0$$

Definition 6.2

$$\mathsf{P} := \bigcup_{p \ Polynom} \mathit{TIME}(p(n))$$

Wir betrachten nur Polynome mit Koeffizienten $a_k, \ldots a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \cdots + a_1 n + a_0$$

Definition 6.2

$$\mathsf{P} := \bigcup_{p \ Polynom} \mathit{TIME}(p(n))$$

Damit gilt auch

$$\mathsf{P} = \bigcup_{k \geq 0} \mathit{TIME}(O(n^k))$$

wobei

$$\mathit{TIME}(O(f)) := \bigcup_{g \in O(f)} \mathit{TIME}(g)$$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G$ ist CFG $\wedge\, w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G$ ist Graph $\wedge\, w$ ist Pfad in $G\} \in \mathsf{P}$
- $\{bin(p) \mid p$ Primzahl$\} \in \mathsf{P}$
- $\{(G, w) \mid G$ ist Graph $\wedge\, w$ ist Eulerkreis in $G\} \in \mathsf{P}$

Beweis durch Angabe eines Algorithmus mit Komplexität $O(n^k)$ für ein $k \geq 1$.

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G$ ist CFG $\wedge\, w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G$ ist Graph $\wedge\, w$ ist Pfad in $G\} \in \mathsf{P}$
- $\{bin(p) \mid p$ Primzahl$\} \in \mathsf{P}$
- $\{(G, w) \mid G$ ist Graph $\wedge\, w$ ist Eulerkreis in $G\} \in \mathsf{P}$

Beweis durch Angabe eines Algorithmus mit Komplexität $O(n^k)$ für ein $k \geq 1$.

Bemerkungen

- $O(n \log n) \subset O(n^2)$
- $n^{\log n}, 2^n \notin O(n^k)$ für alle k

- Warum P und nicht (zB) $O(n^{17})$?

- Warum P und nicht (zB) $O(n^{17})$?
  Um robust bzgl Maschinenmodell zu sein:
  1-Band DTM braucht $O(t^2)$ Schritte
  um t Schritte einer k-Band DTM zu simulieren.

  Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

  Offen: Quantencomputer

- Warum P und nicht (zB) $O(n^{17})$?
  Um robust bzgl Maschinenmodell zu sein:
  1-Band DTM braucht $O(t^2)$ Schritte
  um t Schritte einer k-Band DTM zu simulieren.

  Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

  Offen: Quantencomputer

- Warum TM?

- Warum P und nicht (zB) $O(n^{17})$?
  Um robust bzgl Maschinenmodell zu sein:
  1-Band DTM braucht $O(t^2)$ Schritte
  um t Schritte einer k-Band DTM zu simulieren.

  Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

  Offen: Quantencomputer

- Warum TM?
  Historisch. Ebenfalls möglich: (zB) WHILE.
  Aber zwei mögliche Kostenmodelle:

  Uniform x_i := x_j + n kostet 1 Zeiteinheit.
  Logarithmisch x_i := x_j + n kostet $\log x_j$ Zeiteinheiten.

6.2 Die Komplexitätsklasse NP

Berechnungsmodell:

NTM = nichtdeterministische Mehrband-TM

- NP ist die Klasse der Sprachen, die von einer NTM in polynomieller Zeit akzeptiert werden.
- Dh: Eine Sprache A liegt in NP gdw es ein Polynom $p(n)$ und eine NTM M gibt, so dass
  1. $L(M) = A$ und
  2. für alle $w \in A$ *kann* $M[w]$ in $\leq p(|w|)$ Schritten akzeptieren, dh einen Endzustand erreichen.

Definition 6.4

$$\mathit{ntime}_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} \\ \text{bis NTM } M[w] \text{ akzeptiert} & \text{falls } w \in L(M) \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Definition 6.4

$$\mathit{ntime}_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} \\ \text{bis NTM } M[w] \text{ akzeptiert} & \text{falls } w \in L(M) \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.

$$\begin{aligned} \mathit{NTIME}(f(n)) &:= \{A \subseteq \Sigma^* \mid \exists \text{NTM } M.\ A = L(M) \wedge \\ &\qquad \forall w \in \Sigma^*.\ \mathit{ntime}_M(w) \leq f(|w|)\} \\ \text{NP} &:= \bigcup_{p\ Polynom} \mathit{NTIME}(p(n)) \end{aligned}$$

Definition 6.4

$$\mathit{ntime}_M(w) := \begin{cases} \text{minimale Anzahl der Schritte bis NTM } M[w] \text{ akzeptiert} & \text{falls } w \in L(M) \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.

$$\begin{aligned} \mathit{NTIME}(f(n)) &:= \{A \subseteq \Sigma^* \mid \exists \text{NTM } M.\ A = L(M) \wedge \\ &\qquad \forall w \in \Sigma^*.\ \mathit{ntime}_M(w) \leq f(|w|)\} \\ \text{NP} &:= \bigcup_{p\ Polynom} \mathit{NTIME}(p(n)) \end{aligned}$$

Bemerkungen:

- P $\subseteq$ NP
- Seit etwa 1970 ist offen ob P $=$ NP.

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnunsgfolgen haben.

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnunsgfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnunsgfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

NP′ $\subseteq$ NP: Klar.

NP $\subseteq$ NP′: Falls $A \in$ NP mit Polynom p und NTM M,
so kann man NTM M' konstruieren mit $L(M') = A$,
so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

1. Eingabe w
2. Schreibe $p(|w|)$ auf ein getrenntes Band („timer“)
3. Simuliere $M[w]$, aber dekrementiere timer nach jedem Schritt.
4. Wird timer=0, ohne dass M gehalten hat, halte in einem nicht-Endzustand („timeout“)

Beispiel 6.5

- Ein Euler-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jede Kante genau einmal enthält.
- Satz: Ein Graph hat einen Euler-Kreis gdw er zusammenhängend ist, und jeder Knoten geraden Grad hat.
- Beide Eigenschaften sind in polynomieller Zeit von einer DTM überprüfbar.
- $\Longrightarrow \{G \mid G \text{ hat Euler-Kreis}\} \in \mathsf{P}$

Beispiel 6.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.
- HAMILTON $:= \{G \mid G \text{ hat Hamilton-Kreis}\} \in$ NP
  mit folgendem Algorithmus der Art „Rate und prüfe“:
  1. Rate eine Permutation der Knoten des Graphen.
  2. Prüfe, ob diese Permutation ein Hamilton-Kreis ist.

  Das Raten ist in polynomieller Zeit von einer NTM machbar.
  Das Prüfen ist in polynomieller Zeit von einer DTM machbar.

Vermutung: HAMILTON $\notin$ P, da man keinen substanziell besseren Algorithmus kennt, als alle Permutationen auszuprobieren.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Alle diese Probleme können nichtdeterministisch in polynomieller zeit gelöst werden und sind daher in NP. Wir zeigen nun, dass *alle* Probleme in NP so formuliert werden können.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Alle diese Probleme können nichtdeterministisch in polynomieller zeit gelöst werden und sind daher in NP. Wir zeigen nun, dass *alle* Probleme in NP so formuliert werden können.

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Alle diese Probleme können nichtdeterministisch in polynomieller zeit gelöst werden und sind daher in NP. Wir zeigen nun, dass *alle* Probleme in NP so formuliert werden können.

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*.\ w\#c \in L(M)\}$
  falls es ein Polynom p gibt, so dass $\mathit{time}_M(w\#c) \leq p(|w|)$.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Alle diese Probleme können nichtdeterministisch in polynomieller zeit gelöst werden und sind daher in NP. Wir zeigen nun, dass *alle* Probleme in NP so formuliert werden können.

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*.\ w\#c \in L(M)\}$
  falls es ein Polynom p gibt, so dass $\mathit{time}_M(w\#c) \leq p(|w|)$.

NB:
In Zeit $p(n)$ kann M maximal die ersten $p(n)$ Zeichen von c lesen. Daher genügt für w ein Zertifikat der Länge $\leq p(|w|)$.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Alle diese Probleme können nichtdeterministisch in polynomieller zeit gelöst werden und sind daher in NP. Wir zeigen nun, dass *alle* Probleme in NP so formuliert werden können.

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*.\ w\#c \in L(M)\}$
  falls es ein Polynom p gibt, so dass $\mathit{time}_M(w\#c) \leq p(|w|)$.

NB:
In Zeit $p(n)$ kann M maximal die ersten $p(n)$ Zeichen von c lesen. Daher genügt für w ein Zertifikat der Länge $\leq p(|w|)$.

Beispiel 6.8 (HAMILTON)

Zertifikat: Knotenpermutation. In polynomieller Zeit verifizierbar.

Satz 6.9

$A \in$ NP gdw es gibt einen polynomiell beschränkten Verifikator für A.

Satz 6.9

$A \in$ NP gdw es gibt einen polynomiell beschränkten Verifikator für A.

Beweis:

„$\Rightarrow$“:

Sei $A \in$ NP. Dh es gibt NTM N, die A in Zeit $p(n)$ akzeptiert.
Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnunsgfolge von $N[w]$ mit $\leq p(n)$ Schritten.

Satz 6.9

$A \in$ NP gdw es gibt einen polynomiell beschränkten Verifikator für A.

Beweis:

„$\Rightarrow$“:

Sei $A \in$ NP. Dh es gibt NTM N, die A in Zeit $p(n)$ akzeptiert.
Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnunsgfolge von $N[w]$ mit $\leq p(n)$ Schritten.
Ein polynomiell beschränkter Verfikator für $L(N)$:

1. Eingabe $w\#c$
2. Simuliere $N[w]$, gesteuert durch die Transitionen in c.
3. Überprüfe dabei, ob die Transition in c jeweils zu N und zur augenblicklichen Konfiguration von N passt.
4. Akzeptiere, falls c in einen Endzustand führt.

Beweis (Forts.):

„$\Leftarrow$“:

Sei M ein polynomiell (durch p) beschränkter Verifikator für A.

Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

Beweis (Forts.):

„$\Leftarrow$“:

Sei M ein polynomiell (durch p) beschränkter Verifikator für A.
Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

1. Eingabe w.
2. Schreibe hinter w zuerst $\#$ und dann ein nichtdeterministisch gewähltes Wort $c \in \Delta^*$, $|c| = p(|w|)$:
   for $i := 1, \ldots, p(|w|)$ **do**
   schreibe ein Zeichen aus Δ und gehe nach rechts
3. Führe M aus (mit Eingabe $w\#c$).

Nach Annahme gilt $\mathit{time}_M(w\#c) \leq p(|w|)$.
Damit braucht $N[w]$ $O(p(|w|))$ Schritte. $\square$

Fazit:

Fazit:

P sind die Sprachen, bei denen $w \in L$ schnell entschieden werden kann.

Fazit:

P sind die Sprachen, bei denen $w \in L$ schnell entschieden werden kann.

NP sind die Sprachen, bei denen ein Zertifkat für $w \in L$ schnell verifiziert/überprüft werden kann.

Fazit:

P sind die Sprachen, bei denen $w \in L$ schnell entschieden werden kann.

NP sind die Sprachen, bei denen ein Zertifkat für $w \in L$ schnell verifiziert/überprüft werden kann.

Intuition:
Es ist leichter, eine Lösung zu verifizieren als zu finden.

Fazit:

P sind die Sprachen, bei denen $w \in L$ schnell entschieden werden kann.

NP sind die Sprachen, bei denen ein Zertifkat für $w \in L$ schnell verifiziert/überprüft werden kann.

Intuition:
Es ist leichter, eine Lösung zu verifizieren als zu finden.

Aber:
Noch wurde von keiner Sprache bewiesen, dass sie in NP\P liegt.

6.3 NP-Vollständigkeit

1. Polynomielle Reduzierbarkeit $\leq_p$
2. NP-vollständige Probleme = härteste Probleme in NP,
   alle anderen Probleme in NP darauf polynomiell reduzierbar
3. Satz: SAT ist NP-vollständig

Definition 6.10

Sei $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$.
Dann heißt A polynomiell reduzierbar auf B, $A \leq_p B$,
gdw es eine totale und von einer DTM in polynomieller Zeit berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$

$$w \in A \iff f(w) \in B$$

Definition 6.10

Sei $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$.
Dann heißt A polynomiell reduzierbar auf B, $A \leq_p B$,
gdw es eine totale und von einer DTM in polynomieller Zeit berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$

$$w \in A \iff f(w) \in B$$

Da $q(p(n))$ ein Polynom ist falls p und q Polynome sind:

Lemma 6.11

Die Relation $\leq_p$ ist transitiv.

Beispiel 6.12

HAMILTONKREIS

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“ und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge $\leq k$?

HAMILTON $\leq_p$ TSP

Beispiel 6.12

HAMILTONKREIS

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“ und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge $\leq k$?

HAMILTON $\leq_p$ TSP

$$(\{1, \ldots, n\}, E) \;\mapsto\; (M, n)$$

wobei

$$M_{ij} \;:=\; \begin{cases} 1 & \text{falls } \{i,j\} \in E \\ 2 & \text{sonst} \end{cases}$$

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in \mathit{P/NP} \implies A \in \mathit{P/NP}$$

Beweis:

- Sei $A \leq_p B$ mittels f, die von DTM M_f berechnet wird.
  Polynom p begrenzt Rechenzeit von M_f.
- Sei $B \in$ P mittels DTM M.
  Polynom q begrenzt Rechenzeit von M.

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in \mathit{P/NP} \implies A \in \mathit{P/NP}$$

Beweis:

- Sei $A \leq_p B$ mittels f, die von DTM M_f berechnet wird.
  Polynom p begrenzt Rechenzeit von M_f.
- Sei $B \in$ P mittels DTM M.
  Polynom q begrenzt Rechenzeit von M.

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

- $M_f[w]$ macht $\leq p(|w|)$ Schritte.
- Ausgabe $f(w)$ von M_f hat Länge $\leq |w| + p(|w|)$.
- M macht $\leq q(|f(w)|) \leq q(|w|+p(|w|))$ Schritte (q monoton)

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in \mathit{P/NP} \implies A \in \mathit{P/NP}$$

Beweis:

- Sei $A \leq_p B$ mittels f, die von DTM M_f berechnet wird.
  Polynom p begrenzt Rechenzeit von M_f.
- Sei $B \in$ P mittels DTM M.
  Polynom q begrenzt Rechenzeit von M.

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

- $M_f[w]$ macht $\leq p(|w|)$ Schritte.
- Ausgabe $f(w)$ von M_f hat Länge $\leq |w| + p(|w|)$.
- M macht $\leq q(|f(w)|) \leq q(|w|+p(|w|))$ Schritte (q monoton)

Daher macht $(M_f; M)[w]$ maximal $p(|w|) + q(|w| + p(|w|))$ Schritte, ein Polynom in $|w|$.

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in \textit{P/NP} \implies A \in \textit{P/NP}$$

Beweis:

- Sei $A \leq_p B$ mittels f, die von DTM M_f berechnet wird.
  Polynom p begrenzt Rechenzeit von M_f.
- Sei $B \in$ P mittels DTM M.
  Polynom q begrenzt Rechenzeit von M.

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

- $M_f[w]$ macht $\leq p(|w|)$ Schritte.
- Ausgabe $f(w)$ von M_f hat Länge $\leq |w| + p(|w|)$.
- M macht $\leq q(|f(w)|) \leq q(|w|+p(|w|))$ Schritte (q monoton)

Daher macht $(M_f; M)[w]$ maximal $p(|w|) + q(|w| + p(|w|))$ Schritte, ein Polynom in $|w|$.

Analog: $A \leq_p B \in$ NP $\implies A \in$ NP $\square$

Ein Problem ist NP-schwer,
wenn es mindestens so hart wie alles in NP ist:

Ein Problem ist NP-schwer,
wenn es mindestens so hart wie alles in NP ist:

Definition 6.14

Eine Sprache L heißt NP-schwer gdw $A \leq_p L$ für alle $A \in$ NP.

Ein Problem ist NP-schwer,
wenn es mindestens so hart wie alles in NP ist:

Definition 6.14

Eine Sprache L heißt NP-schwer gdw $A \leq_p L$ für alle $A \in$ NP.

Definition 6.15

Eine Sprache L heißt NP-vollständig gdw
L NP-schwer ist und $L \in$ NP.

Ein Problem ist NP-schwer,
wenn es mindestens so hart wie alles in NP ist:

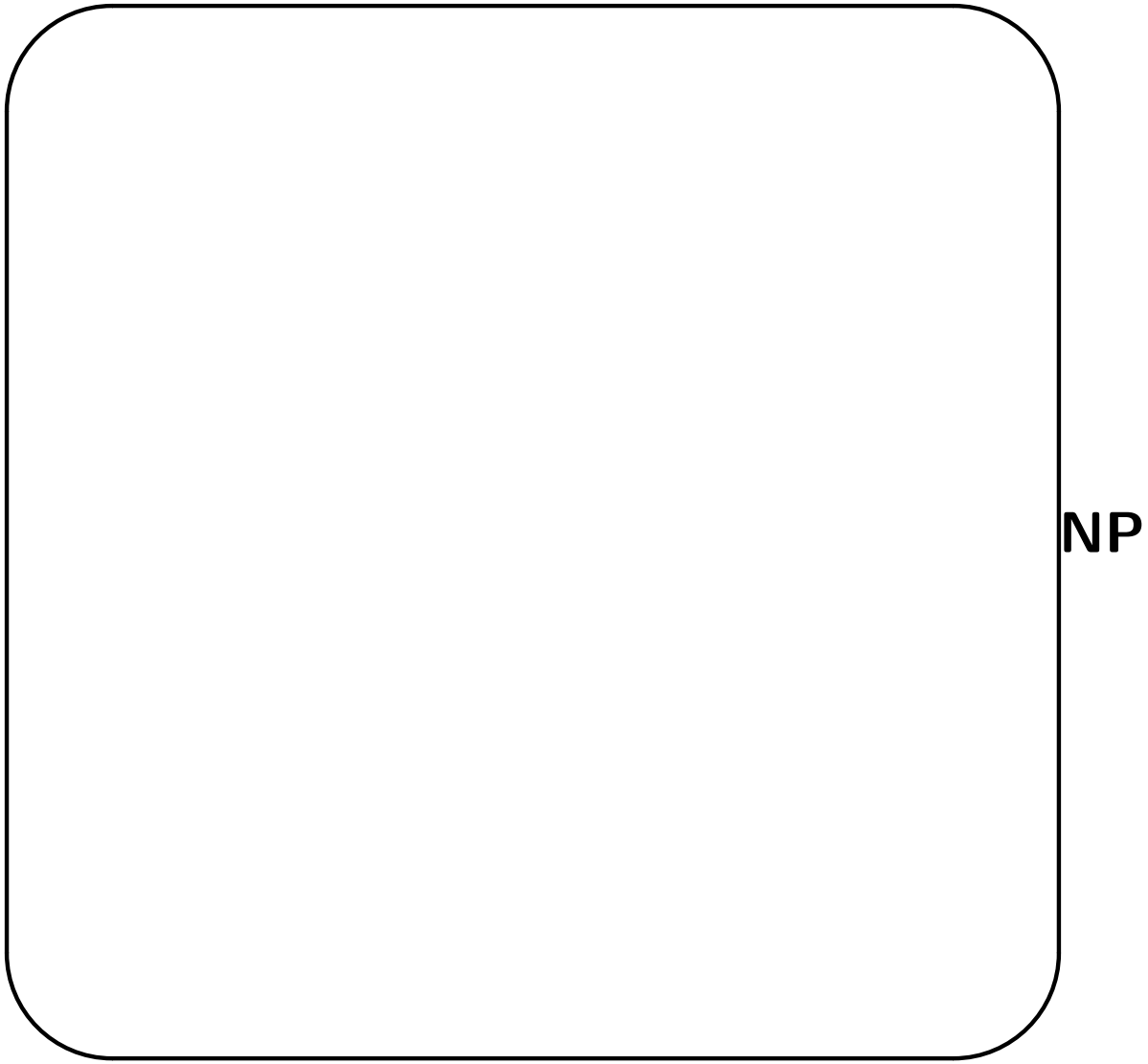
Definition 6.14

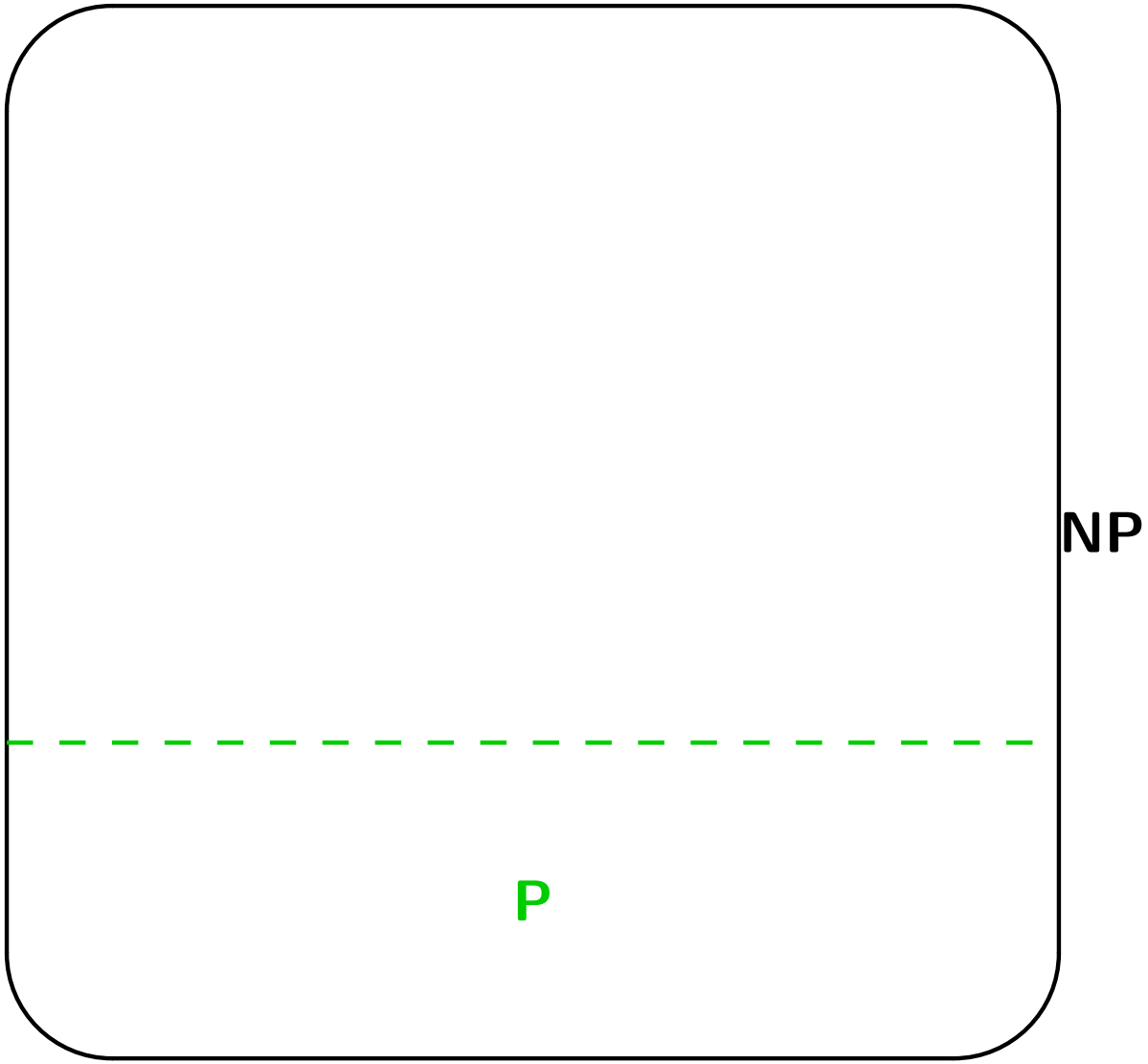
Eine Sprache L heißt NP-schwer gdw $A \leq_p L$ für alle $A \in$ NP.

Definition 6.15

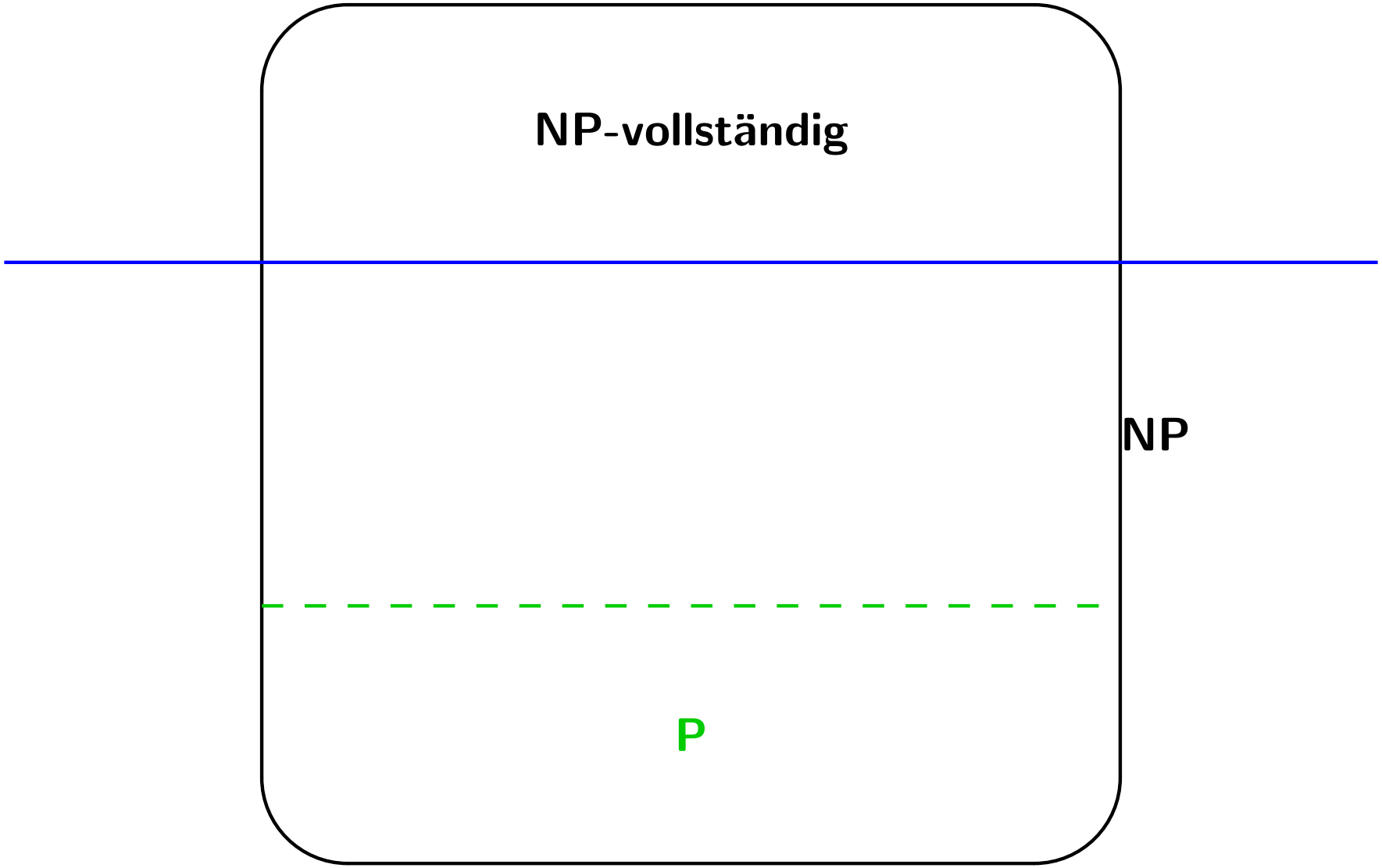
Eine Sprache L heißt NP-vollständig gdw
L NP-schwer ist und $L \in$ NP.

Intuition: NP-vollständige Probleme sind die schwierigsten Probleme in NP:
alle Probleme in NP sind polynomiell auf sie reduzierbar.


NP


NP
P

NP-schwer



Wie man $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ lösen kann:

Lemma 6.16

Es gilt P=NP gdw ein NP-vollständiges Problem in P liegt.

Wie man $P \stackrel{?}{=} NP$ lösen kann:

Lemma 6.16

Es gilt P=NP gdw ein NP-vollständiges Problem in P liegt.

Beweis:

„$\Rightarrow$“: Falls P=NP, so liegt jedes NP-vollständige Problem in P.

Wie man P$\stackrel{?}{=}$NP lösen kann:

Lemma 6.16

Es gilt P=NP gdw ein NP-vollständiges Problem in P liegt.

Beweis:

„$\Rightarrow$“: Falls P=NP, so liegt jedes NP-vollständige Problem in P.

„$\Leftarrow$“: Sei L ein NP-vollständiges Problem in P. Dann gilt P $\supseteq$ NP: Ist $A \in$ NP, so gilt $A \leq_p L$ (da L NP-schwer) und nach Lemma 6.13 $A \in$ P (da $L \in$ P). $\square$

Wie man P$\stackrel{?}{=}$NP lösen kann:

Lemma 6.16

Es gilt P=NP gdw ein NP-vollständiges Problem in P liegt.

Beweis:

„$\Rightarrow$“: Falls P=NP, so liegt jedes NP-vollständige Problem in P.

„$\Leftarrow$“: Sei L ein NP-vollständiges Problem in P. Dann gilt P $\supseteq$ NP: Ist $A \in$ NP, so gilt $A \leq_p L$ (da L NP-schwer) und nach Lemma 6.13 $A \in$ P (da $L \in$ P). $\square$

Starke Vermutung:

- P $\neq$ NP
- dh kein NP-vollständiges Problem ist in P.

Wie man P$\stackrel{?}{=}$NP lösen kann:

Lemma 6.16

Es gilt P=NP gdw ein NP-vollständiges Problem in P liegt.

Beweis:

„$\Rightarrow$“: Falls P=NP, so liegt jedes NP-vollständige Problem in P.

„$\Leftarrow$“: Sei L ein NP-vollständiges Problem in P. Dann gilt P $\supseteq$ NP: Ist $A \in$ NP, so gilt $A \leq_p L$ (da L NP-schwer) und nach Lemma 6.13 $A \in$ P (da $L \in$ P). $\square$

Starke Vermutung:

- P $\neq$ NP
- dh kein NP-vollständiges Problem ist in P.

Aber gibt es überhaupt NP-vollständige Probleme?

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{array}{rcl} \text{Variablen:} \quad X & \rightarrow & x \mid y \mid z \mid \ldots \\ \text{Formeln:} \quad F & \rightarrow & X \mid \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \end{array}$$

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{array}{rcl} \text{Variablen:} \quad X & \rightarrow & x \mid y \mid z \mid \ldots \\ \text{Formeln:} \quad F & \rightarrow & X \mid \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \end{array}$$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf einige Klammern weglassen

- Außerste Klammer: $(x \vee y) \wedge z$ statt $((x \vee y) \wedge z)$
- Assoziativität: $(x \vee y \vee z) \wedge \neg x$ statt $((x \vee y) \vee z) \wedge \neg x$.

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{array}{rcl} \text{Variablen:} \quad X & \rightarrow & x \mid y \mid z \mid \ldots \\ \text{Formeln:} \quad F & \rightarrow & X \mid \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \end{array}$$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf einige Klammern weglassen

- Außerste Klammer: $(x \vee y) \wedge z$ statt $((x \vee y) \wedge z)$
- Assoziativität: $(x \vee y \vee z) \wedge \neg x$ statt $((x \vee y) \vee z) \wedge \neg x$.

Semantik der Aussagenlogik:

- Eine Belegung ist eine Funktion von Variablen auf $\{0, 1\}$.
  Bsp: $\sigma = \{x \mapsto 0, y \mapsto 1, z \mapsto 0, \ldots\}$
- Belegungen werden mittels Wahrheitstabellen auf Formeln erweitert. Bsp: $\sigma((\neg x \wedge y) \vee (x \wedge \neg z)) = 1$
- Eine Formel F ist erfüllbar gdw es eine Belegung σ gibt mit $\sigma(F) = 1$.

SAT

Gegeben: Eine aussagenlogische Formel F.

Problem: Ist F erfüllbar?

SAT

Gegeben: Eine aussagenlogische Formel F.

Problem: Ist F erfüllbar?

Lemma 6.17

SAT $\in$ *NP*

Beweis:

Belegungen sind Zertifikate, die in polynomieller Zeit geprüft werden können: Es gibt eine DTM, die bei Eingabe einer Formel F und einer Belegung σ für die Variablen in F, in polynomieller Zeit $\sigma(F)$ berechnet. $\square$

Satz 6.18 (Cook 1971)

SAT ist NP-vollständig.

Satz 6.18 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-schwer.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.

Satz 6.18 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-schwer.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.

Da $A \in$ NP gibt es NTM M mit $A = L(M)$ und
Polynom p mit *ntime*$_M(w) \leq p(|w|)$.

Reduktion bildet $w = x_1 \ldots x_n \in \Sigma^*$ auf eine Formel F ab.

- In polynomieller Zeit.
- So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Beweis (Forts.):

Die Variablen von F beschreiben das *mögliche* Verhalten von $M[w]$:

$\mathit{zust}_{t,q}$	$t = 0, \ldots, p(n)$ $q \in Q$	$\mathit{zust}_{t,q} = 1 \;\Leftrightarrow$ Zustand nach t Schritten ist q
$\mathit{pos}_{t,i}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots p(n)$	$\mathit{pos}_{t,i} = 1 \;\Leftrightarrow$ Kopfposition nach t Schritten ist i
$\mathit{band}_{t,i,a}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots, p(n)$ $a \in \Gamma$	$\mathit{band}_{t,i,a} = 1 \;\Leftrightarrow$ Bandinhalt nach t Schritten auf Bandposition i ist Zeichen a

Beweis (Forts.):

Die Variablen von F beschreiben das *mögliche* Verhalten von $M[w]$:

$zust_{t,q}$	$t = 0, \ldots, p(n)$ $q \in Q$	$zust_{t,q} = 1 \;\Leftrightarrow$ Zustand nach t Schritten ist q
$pos_{t,i}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots p(n)$	$pos_{t,i} = 1 \;\Leftrightarrow$ Kopfposition nach t Schritten ist i
$band_{t,i,a}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots, p(n)$ $a \in \Gamma$	$band_{t,i,a} = 1 \;\Leftrightarrow$ Bandinhalt nach t Schritten auf Bandposition i ist Zeichen a

Berechnung von M auf w	$\rightarrow$	Belegung
Belegung	$\not\rightarrow$	Berechnung von M auf w

Beweis (Forts.):

Die Variablen von F beschreiben das *mögliche* Verhalten von $M[w]$:

$zust_{t,q}$	$t = 0, \ldots, p(n)$ $q \in Q$	$zust_{t,q} = 1 \;\Leftrightarrow$ Zustand nach t Schritten ist q
$pos_{t,i}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots p(n)$	$pos_{t,i} = 1 \;\Leftrightarrow$ Kopfposition nach t Schritten ist i
$band_{t,i,a}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots, p(n)$ $a \in \Gamma$	$band_{t,i,a} = 1 \;\Leftrightarrow$ Bandinhalt nach t Schritten auf Bandposition i ist Zeichen a

Berechnung von M auf w	$\rightarrow$	Belegung
Belegung	$\not\rightarrow$	Berechnung von M auf w

Idee: Finde $F = F_1 \wedge F_2$ mit

Erfüllende Belegung von F_1	$\rightarrow$	Berechnung von M auf w
Erfüllende Belegung von $F_1 \wedge F_2$	$\rightarrow$	Akzeptierende Berechnung von M auf w

Beweis (Forts.):

Sei $Q = \{q_1, \ldots, q_k\}$ und $\Gamma = \{a_1, \ldots, a_l\}$.

$$F \quad := \quad R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

Hilfsformel : $\quad G(v_1, \ldots, v_r) := (\bigvee_{i=1}^{r} v_i) \wedge \left(\bigwedge_{i=1}^{r-1} \bigwedge_{j=i+1}^{r} \neg(v_i \wedge v_j) \right)$

Beweis (Forts.):

Sei $Q = \{q_1, \ldots, q_k\}$ und $\Gamma = \{a_1, \ldots, a_l\}$.

$$F \quad := \quad R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

Hilfsformel : $\quad G(v_1, \ldots, v_r) := (\bigvee_{i=1}^{r} v_i) \wedge \left(\bigwedge_{i=1}^{r-1} \bigwedge_{j=i+1}^{r} \neg(v_i \wedge v_j) \right)$

Belegung erfüllt $R \quad \rightarrow \quad$ "Ein Zustand, eine Position, und ein Zeichen pro Feld"

$$R \quad := \quad \bigwedge_t [G(\mathit{zust}_{t,q_1}, \ldots, \mathit{zust}_{t,q_k}) \wedge G(\mathit{pos}_{t,-p(n)}, \ldots, \mathit{pos}_{t,p(n)}) \wedge \bigwedge_i G(\mathit{band}_{t,i,a_1}, \ldots, \mathit{band}_{t,i,a_l})]$$

Beweis (Forts.):

Belegung erfüllt A $\rightarrow$ "Berechnung startet aus der Anfangskonfiguration von w"

$$A := \mathit{zust}_{0,q_1} \wedge \mathit{pos}_{0,1} \wedge \bigwedge_{j=1}^{n} \mathit{band}_{0,j,x_j} \wedge \bigwedge_{j=-p(n)}^{0} \mathit{band}_{0,j,\Box} \wedge \bigwedge_{j=n+1}^{p(n)} \mathit{band}_{0,j,\Box}$$

Beweis (Forts.):

Belegung erfüllt $T_1 \wedge T_2 \quad \rightarrow \quad$ "Berechnung respektiert die Übergangsrelation von M"

$$
\begin{aligned}
T_1 \quad := \quad & \bigwedge_{t,q,i,a} [\mathit{zust}_{t,q} \wedge \mathit{pos}_{t,i} \wedge \mathit{band}_{t,i,a} \\
& \rightarrow \bigvee_{(q',a',y) \in \delta(q,a)} (\mathit{zust}_{t+1,q'} \wedge \mathit{pos}_{t+1,i+y} \wedge \mathit{band}_{t+1,i,a'})] \\
T_2 \quad := \quad & \bigwedge_{t,i,a} ((\neg \mathit{pos}_{t,i} \wedge \mathit{band}_{t,i,a}) \rightarrow \mathit{band}_{t+1,i,a})
\end{aligned}
$$

Beweis (Forts.):

Belegung erfüllt $T_1 \wedge T_2$ $\rightarrow$ "Berechnung respektiert die Übergangsrelation von M"

$$T_1 := \bigwedge_{t,q,i,a} [\mathit{zust}_{t,q} \wedge \mathit{pos}_{t,i} \wedge \mathit{band}_{t,i,a} \rightarrow \bigvee_{(q',a',y)\in\delta(q,a)} (\mathit{zust}_{t+1,q'} \wedge \mathit{pos}_{t+1,i+y} \wedge \mathit{band}_{t+1,i,a'})]$$

$$T_2 := \bigwedge_{t,i,a} ((\neg \mathit{pos}_{t,i} \wedge \mathit{band}_{t,i,a}) \rightarrow \mathit{band}_{t+1,i,a})$$

Belegung erfüllt E $\rightarrow$ "Berechnung erreicht eine Konfiguration mit Endzustand"

$$E := \bigvee_{t} \bigvee_{q\in F} \mathit{zust}_{t,q}$$

Beweis (Forts.):

Mit $|Q| = k$, $|\Gamma| = l$, $|w| = n$:

Länge von R:	$O\big(p(n) \cdot \big((p(n) \cdot k)^2 + (p(n)^2)^2 + (p(n)^2 \cdot l)^2\big)\big)$
Länge von A:	$O(p(n))$
Länge von $T_1 \wedge T_2$:	$O(p(n)^2 \cdot k^2 \cdot l^2)$
Länge von E:	$O(p(n) \cdot k)$

$\square$

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT "reduziert" werden

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT "reduziert" werden

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")
else
  for $i := 1$ **to** k **do**
    if $F[x_i := 0] \in$ SAT **then** $b := 0$ **else** $b := 1$
    output(x_i "=" b)
    $F := F[x_i := b]$

wobei $F[x := b] \;=\; F$ mit x ersetzt durch b.

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT "reduziert" werden

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")
else
  for $i := 1$ **to** k **do**
    if $F[x_i := 0] \in$ SAT **then** $b := 0$ **else** $b := 1$
    output(x_i "=" b)
    $F := F[x_i := b]$

wobei $F[x := b] \;=\; F$ mit x ersetzt durch b.

Entscheidung von SAT in Zeit $O(f(n))$
$\Longrightarrow$ Berechnung einer erf. Bel. in Zeit $O(n \cdot (f(n) + n))$
(falls es eine gibt.)

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT "reduziert" werden

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")
else
  for $i := 1$ **to** k **do**
    if $F[x_i := 0] \in$ SAT **then** $b := 0$ **else** $b := 1$
    output(x_i "=" b)
    $F := F[x_i := b]$

wobei $F[x := b] \;=\; F$ mit x ersetzt durch b.

Entscheidung von SAT in Zeit $O(f(n))$
$\Longrightarrow$ Berechnung einer erf. Bel. in Zeit $O(n \cdot (f(n) + n))$
(falls es eine gibt.)

$$f(n) \text{ polynomiell} \implies n \cdot (f(n) + n) \text{ polynomiell}$$
$$f(n) \text{ exponentiell} \implies n \cdot (f(n) + n) \text{ exponentiell}$$

Bemerkungen:

Bemerkungen:

- Die Reduktion der Lösungsberechnung auf SAT ist eine rein theoretische Konstruktion.
- Sie zeigt, dass man sich auf SAT beschränken kann, wenn man nur an polynomiell/exponentiell interessiert ist.
- Alle bekannten Entscheidungsverfahren für SAT berechnen auch eine Lösung.

Von NP-schwer zu „NP-leicht“

- Bis vor ca. 15 Jahren:

$$\text{NP-vollständig} = \text{Todesurteil}$$

Von NP-schwer zu „NP-leicht“

- Bis vor ca. 15 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 15 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von SAT-Lösern (*SAT-solver*): `http://satcompetition.org`

Von NP-schwer zu „NP-leicht“

- Bis vor ca. 15 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 15 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von
  SAT-Lösern (*SAT-solver*): `http://satcompetition.org`
  Stand der Kunst: Erfüllbar: bis 10^5 Variablen

Von NP-schwer zu „NP-leicht“

- Bis vor ca. 15 Jahren:

$$\text{NP-vollständig} = \text{Todesurteil}$$

- In den letzten 15 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von SAT-Lösern (*SAT-solver*): `http://satcompetition.org`
  Stand der Kunst: Erfüllbar: bis 10^5 Variablen
  Unerfüllbar: bis 10^3 Variablen

Von NP-schwer zu „NP-leicht“

- Bis vor ca. 15 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 15 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von SAT-Lösern (*SAT-solver*): `http://satcompetition.org`
  Stand der Kunst: Erfüllbar: bis 10^5 Variablen
  Unerfüllbar: bis 10^3 Variablen

- Jetzt:

  NP-vollständig = Hoffnung durch SAT

Von NP-schwer zu „NP-leicht“

- Bis vor ca. 15 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 15 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von SAT-Lösern (*SAT-solver*): `http://satcompetition.org`
  Stand der Kunst: Erfüllbar: bis 10^5 Variablen
  Unerfüllbar: bis 10^3 Variablen

- Jetzt:

  NP-vollständig = Hoffnung durch SAT

- Paradigma:

  SAT (Logik!) als universelle Sprache
  zur Kodierung kombinatorischer Probleme

  Reduktion auf SAT manchmal schneller als problemspezifische Löser! Und fast immer einfacher.

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen $= V \times \{1, 2, 3\}$. Notatation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen $= V \times \{1, 2, 3\}$. Notatation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Formel:

$$\bigwedge_{v \in V} G(x_{v1}, x_{v2}, x_{v3}) \wedge \bigwedge_{(u,v) \in E} \neg(x_{u1} \wedge x_{v1} \vee x_{u2} \wedge x_{v2} \vee x_{u3} \wedge x_{v3})$$

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen $= V \times \{1, 2, 3\}$. Notatation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Formel:

$$\bigwedge_{v \in V} G(x_{v1}, x_{v2}, x_{v3}) \wedge \bigwedge_{(u,v) \in E} \neg(x_{u1} \wedge x_{v1} \vee x_{u2} \wedge x_{v2} \vee x_{u3} \wedge x_{v3})$$

Bemerkungen

- Zeigt 3COL $\leq_p$ SAT und damit 3COL $\in$ NP.
- Zeigt nicht, dass 3COL NP-vollständig ist.

Industrielle Anwendungen von SAT

1. Äquivalenztest von Schaltkreisen.

Schaltung	$\rightarrow$	Boole'sche Formel
Eingabe	$\rightarrow$	Belegung
Eingabe mit 1-Ergebnis	$\rightarrow$	Erfüllende Belegung
Funktionale Äquivalenz	$\rightarrow$	Logische Äquivalenz

Industrielle Anwendungen von SAT

1. Äquivalenztest von Schaltkreisen.

Schaltung	$\rightarrow$	Boole'sche Formel
Eingabe	$\rightarrow$	Belegung
Eingabe mit 1-Ergebnis	$\rightarrow$	Erfüllende Belegung
Funktionale Äquivalenz	$\rightarrow$	Logische Äquivalenz

NICHTÄQUIVALENZ

Gegeben: Zwei aussagenlogische Formeln F_1, F_2 über Variablenmengen X_1, X_2.

Problem: Gibt es eine Belegung σ von $X_1 \cup X_2$ mit $\sigma(F_1)\neg\sigma(F_2)$?

Lemma 6.19

NICHTÄQUIVALENZ $\leq_p$ *SAT und SAT* $\leq_p$ *NICHTÄQUIVALENZ.*

Lemma 6.19
NICHTÄQUIVALENZ $\leq_p$ *SAT und SAT* $\leq_p$ *NICHTÄQUIVALENZ.*

Beweis:
NICHTÄQUIVALENZ $\leq_p$ SAT:

Lemma 6.19

NICHTÄQUIVALENZ $\leq_p$ *SAT und SAT* $\leq_p$ *NICHTÄQUIVALENZ.*

Beweis:

NICHTÄQUIVALENZ $\leq_p$ SAT:

$$f(F_1, F_2) = (F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$$

Lemma 6.19
NICHTÄQUIVALENZ $\leq_p$ SAT und SAT $\leq_p$ NICHTÄQUIVALENZ.

Beweis:
NICHTÄQUIVALENZ $\leq_p$ SAT:

$$f(F_1, F_2) = (F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$$

SAT $\leq_p$ NICHTÄQUIVALENZ:

Lemma 6.19
NICHTÄQUIVALENZ $\leq_p$ SAT und SAT $\leq_p$ NICHTÄQUIVALENZ.

Beweis:
NICHTÄQUIVALENZ $\leq_p$ SAT:

$$f(F_1, F_2) = (F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$$

SAT $\leq_p$ NICHTÄQUIVALENZ:

$$f(F) = (F, x \wedge \neg x)$$

$\square$

2. Bounded Model Checking

Hardware Entscheide, ob eine Schaltung mit Zustand für alle Eingaben innerhalb von 10 Zyklen ein bestimmtes Verhalten (nicht) hat.

2. Bounded Model Checking

Hardware Entscheide, ob eine Schaltung mit Zustand für alle Eingaben innerhalb von 10 Zyklen ein bestimmtes Verhalten (nicht) hat.

Software Entscheide, ob ein Programm für alle Eingaben innerhalb von 10 Schritten ein bestimmtes Verhalten (nicht) hat. Variablen müssen auf sehr kleine Wertebereiche eingeschränkt werden!

3. Programmoptimierung. Beispiel: Registerverteilung

> *Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?*

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

3. Programmoptimierung. Beispiel: Registerverteilung

Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k-Färbbarkeit (und damit auf SAT):

Knoten	$=$	Variable
u und v verbunden	$=$	$u \neq v$ und es gibt einen Programmpunkt, an dem u und v lebendig sind
Farbe	$=$	Register
k-Färbung	$=$	Zuordnung eines Registers zu jeder Variablen

3. Programmoptimierung. Beispiel: Registerverteilung

Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k-Färbbarkeit (und damit auf SAT):

Knoten	$=$	Variable
u und v verbunden	$=$	$u \neq v$ und es gibt einen Programmpunkt, an dem u und v lebendig sind
Farbe	$=$	Register
k-Färbung	$=$	Zuordnung eines Registers zu jeder Variablen

Sowohl k-Färbbarkeit als auch Registerverteilung ist für $k \geq 3$ NP-vollständig.

Mehr Information: Vorlesung *Programmoptimierung*

6.4 Weitere NP-vollständige Probleme

Wie zeigt man, dass ein (weiteres) Problem B NP-vollständig ist?

- Zeige $B \in$ NP (meist trivial)
- Zeige $A \leq_p B$ für ein NP-vollständiges Problem A.

6.4 Weitere NP-vollständige Probleme

Wie zeigt man, dass ein (weiteres) Problem B NP-vollständig ist?

- Zeige $B \in$ NP (meist trivial)
- Zeige $A \leq_p B$ für ein NP-vollständiges Problem A.

Lemma 6.20

Ist A NP-vollständig, so ist $B \in$ NP ebenfalls NP-vollständig, falls $A \leq_p B$.

Beweis:

Folgt direkt aus der Transitivität von $\leq_p$:
B ist NP-schwer, denn für $C \in$ NP gilt $C \leq_p A \leq_p B$. $\square$

Warum will man wissen, dass ein Problem B NP-vollständig ist?

Warum will man wissen, dass ein Problem B NP-vollständig ist?
Um sicher zu sein, dass

- ein polynomieller Algorithmus für B ein Durchbruch wäre
- und daher wahrscheinlich nicht existiert.

Warum will man wissen, dass ein Problem B NP-vollständig ist?
Um sicher zu sein, dass

- ein polynomieller Algorithmus für B ein Durchbruch wäre
- und daher wahrscheinlich nicht existiert.

Praktische Instanzen von B könnten trotzdem (zB mit SAT) „effizient“ lösbar sein.

Viele Varianten von SAT sind ebenfalls NP-vollständig.

Definition 6.21

- Eine Formel ist in Konjunktiver Normalform (KNF) gdw sie eine Konjunktion von Klauseln ist: $K_1 \wedge \cdots \wedge K_n$
- Eine Klausel ist eine Disjunktion von Literalen: $L_1 \vee \cdots \vee L_m$
- Ein Literal ist eine (evtl. negierte) Variable.
- Eine Formel ist in 3KNF gdw jede Klausel ≤ 3 Literale enthält.

Viele Varianten von SAT sind ebenfalls NP-vollständig.

Definition 6.21

- Eine Formel ist in Konjunktiver Normalform (KNF) gdw sie eine Konjunktion von Klauseln ist: $K_1 \wedge \cdots \wedge K_n$
- Eine Klausel ist eine Disjunktion von Literalen: $L_1 \vee \cdots \vee L_m$
- Ein Literal ist eine (evtl. negierte) Variable.
- Eine Formel ist in 3KNF gdw jede Klausel ≤ 3 Literale enthält.

Dh eine KNF ist ein Konjunktion von Disjunktionen von (evtl negierten) Variablen.
Bsp: $(x_9 \vee \neg x_2) \wedge (\neg x_7 \vee x_1 \vee x_6)$

Viele Varianten von SAT sind ebenfalls NP-vollständig.

Definition 6.21

- Eine Formel ist in Konjunktiver Normalform (KNF) gdw sie eine Konjunktion von Klauseln ist: $K_1 \wedge \cdots \wedge K_n$
- Eine Klausel ist eine Disjunktion von Literalen: $L_1 \vee \cdots \vee L_m$
- Ein Literal ist eine (evtl. negierte) Variable.
- Eine Formel ist in 3KNF gdw jede Klausel ≤ 3 Literale enthält.

Dh eine KNF ist ein Konjunktion von Disjunktionen von (evtl negierten) Variablen.
Bsp: $(x_9 \vee \neg x_2) \wedge (\neg x_7 \vee x_1 \vee x_6)$

3KNF-SAT

Gegeben: Ein Formel in 3KNF

Problem: Ist die Formel erfüllbar?

Viele Varianten von SAT sind ebenfalls NP-vollständig.

Definition 6.21

- Eine Formel ist in Konjunktiver Normalform (KNF) gdw sie eine Konjunktion von Klauseln ist: $K_1 \wedge \cdots \wedge K_n$
- Eine Klausel ist eine Disjunktion von Literalen: $L_1 \vee \cdots \vee L_m$
- Ein Literal ist eine (evtl. negierte) Variable.
- Eine Formel ist in 3KNF gdw jede Klausel ≤ 3 Literale enthält.

Dh eine KNF ist ein Konjunktion von Disjunktionen von (evtl negierten) Variablen.
Bsp: $(x_9 \vee \neg x_2) \wedge (\neg x_7 \vee x_1 \vee x_6)$

3KNF-SAT

Gegeben: Ein Formel in 3KNF

Problem: Ist die Formel erfüllbar?

Offensichtlich gilt 3KNF-SAT $\in$ NP.
Aber vielleicht ist 3KNF-SAT einfacher als SAT?

Satz 6.22

3KNF-SAT ist NP-vollständig.

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen SAT $\leq_p$ 3KNF-SAT mit einer polynomiellen Reduktion $F \mapsto F'$ so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar } \Leftrightarrow F' \text{ ist erfüllbar}$$

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen SAT $\leq_p$ 3KNF-SAT mit einer polynomiellen Reduktion $F \mapsto F'$ so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar } \Leftrightarrow F' \text{ ist erfüllbar}$$

NB F und F' sind erfüllbarkeitsäquivalent, aber nicht notwendigerweise auch äquivalent.

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen SAT $\leq_p$ 3KNF-SAT mit einer polynomiellen Reduktion $F \mapsto F'$ so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar } \Leftrightarrow F' \text{ ist erfüllbar}$$

NB F und F' sind erfüllbarkeitsäquivalent, aber nicht notwendigerweise auch äquivalent.

1. Transformiere F in Negations-Normalform (NNF) durch fortgesetze Anwendung der de Morganschen Gesetze

$$\begin{aligned} \neg(A \wedge B) &= \neg A \vee \neg B \\ \neg(A \vee B) &= \neg A \wedge \neg B \\ \neg\neg A &= A \end{aligned}$$

von links nach rechts. F_1 ist Resultat.

Beweis (Forts.):

Für F_1 gilt: $\neg$ nur noch direkt vor Variablen.

2. Betrachte F_1 als Baum, wobei die Literale Blätter sind.
Ordne jedem inneren Knoten eine neue Variable $\in \{y_0, y_1, \dots\}$ zu.
Ordne dabei der Wurzel von F_1 die Variable y_0 zu.

Beweis (Forts.):

Für F_1 gilt: $\neg$ nur noch direkt vor Variablen.

2. Betrachte F_1 als Baum, wobei die Literale Blätter sind.
Ordne jedem inneren Knoten eine neue Variable $\in \{y_0, y_1, \dots\}$ zu.
Ordne dabei der Wurzel von F_1 die Variable y_0 zu.

3. Betrachte die y_i als Abkürzung für die Teilbäume, an deren Wurzeln sie stehen

$$y_i = \begin{array}{ccc} & \circ_i & \\ / & & \backslash \\ l_i & & r_i \end{array}$$

wobei $\circ_i \in \{\wedge, \vee\}$

Beweis (Forts.):

Für F_1 gilt: $\neg$ nur noch direkt vor Variablen.

2. Betrachte F_1 als Baum, wobei die Literale Blätter sind.
Ordne jedem inneren Knoten eine neue Variable $\in \{y_0, y_1, \dots\}$ zu.
Ordne dabei der Wurzel von F_1 die Variable y_0 zu.

3. Betrachte die y_i als Abkürzung für die Teilbäume, an deren Wurzeln sie stehen

$$y_i = \begin{array}{ccc} & \circ_i & \\ / & & \backslash \\ l_i & & r_i \end{array}$$

wobei $\circ_i \in \{\wedge, \vee\}$ und l_i, r_i ein Literal oder eine Variable y_j ist.
Beschreibe F_1 Knoten für Knoten:

$$y_0 \wedge (y_0 \leftrightarrow (l_0 \circ_0 r_0)) \wedge (y_1 \leftrightarrow (l_1 \circ_1 r_1)) \cdots =: F_2$$

Beweis (Forts.):

F_1 erf. $\implies$ F_2 erf.: y_i bekommt Wert seines Teilbaums.
F_2 erf. $\implies$ F_1 erf.: klar

Beweis (Forts.):

F_1 erf. $\implies$ F_2 erf.: y_i bekommt Wert seines Teilbaums.
F_2 erf. $\implies$ F_1 erf.: klar

4. Transformiere jede Äquivalenz in 3KNF:

$$\begin{aligned}
(a \leftrightarrow (b \vee c)) &\mapsto (a \vee \neg b) \wedge (a \vee \neg c) \wedge (\neg a \vee b \vee c) \\
(a \leftrightarrow (b \wedge c)) &\mapsto (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)
\end{aligned}$$

Ergebnis ist F'.

Beweis (Forts.):

F_1 erf. $\implies$ F_2 erf.: y_i bekommt Wert seines Teilbaums.
F_2 erf. $\implies$ F_1 erf.: klar

4. Transformiere jede Äquivalenz in 3KNF:

$$\begin{aligned}(a \leftrightarrow (b \vee c)) &\mapsto (a \vee \neg b) \wedge (a \vee \neg c) \wedge (\neg a \vee b \vee c)\\ (a \leftrightarrow (b \wedge c)) &\mapsto (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)\end{aligned}$$

Ergebnis ist F'.

Jeder Schritt ist in polynomieller Zeit in $|F|$ berechenbar.
Bei Transformation in NNF nimmt mit jedem Schritt

$$\text{Summe der } |G| \text{ für alle Teilformeln } \neg G \text{ von } F$$

ab. Daher erreicht man die NNF in $\leq |F|^2$ Schritten. $\square$

Da jede Formel in 3KNF auch in KNF ist:

Korollar 6.23

KNF-SAT ist NP-vollständig.

Da jede Formel in 3KNF auch in KNF ist:

Korollar 6.23
KNF-SAT ist NP-vollständig.

Kann man wie folgt die NP-Vollständigkeit von KNF-SAT zeigen?

Man zeigt SAT $\leq_p$ KNF-SAT
indem man jede Formel in KNF transformiert.

Da jede Formel in 3KNF auch in KNF ist:

Korollar 6.23
KNF-SAT ist NP-vollständig.

Kann man wie folgt die NP-Vollständigkeit von KNF-SAT zeigen?

Man zeigt SAT $\leq_p$ KNF-SAT
indem man jede Formel in KNF transformiert.

Satz 6.24
2KNF-SAT $\in$ *P*

Ohne Beweis

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \ldots, i_k \in \{1, \ldots, n\}$ mit $M = T_{i_1} \cup \cdots \cup T_{i_k}$?

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \ldots, i_k \in \{1, \ldots, n\}$ mit $M = T_{i_1} \cup \cdots \cup T_{i_k}$?

Beispiel 6.25

$$\begin{array}{lclclcl} T_1 & = & \{1,2\} & & T_2 & = & \{1,3\} \\ T_3 & = & \{3,4\} & & T_4 & = & \{3,5\} \\ M & = & \{1,2,3,4,5\} & & k & = & 3 \end{array}$$

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \ldots, i_k \in \{1, \ldots, n\}$ mit $M = T_{i_1} \cup \cdots \cup T_{i_k}$?

Beispiel 6.25

$$\begin{array}{rclrcl} T_1 & = & \{1,2\} & T_2 & = & \{1,3\} \\ T_3 & = & \{3,4\} & T_4 & = & \{3,5\} \\ M & = & \{1,2,3,4,5\} & k & = & 3 \end{array}$$

Anwendung: Zulieferer

M Menge der Teile, die eine Firma einkaufen muss

T_i Menge der Teile, die Zulieferer i anbietet

Kann die Firma ihre Bedürfnisse mit k Zulieferern abdecken?

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \ldots, i_k \in \{1, \ldots, n\}$ mit $M = T_{i_1} \cup \cdots \cup T_{i_k}$?

Beispiel 6.25

$$\begin{array}{lclcl} T_1 & = & \{1,2\} & \quad T_2 & = & \{1,3\} \\ T_3 & = & \{3,4\} & \quad T_4 & = & \{3,5\} \\ M & = & \{1,2,3,4,5\} & \quad k & = & 3 \end{array}$$

Anwendung: Zulieferer

M Menge der Teile, die eine Firma einkaufen muss

T_i Menge der Teile, die Zulieferer i anbietet

Kann die Firma ihre Bedürfnisse mit k Zulieferern abdecken?

Fakt 6.26

MÜ $\in$ *NP.*

Satz 6.27

MÜ ist NP-vollständig.

Satz 6.27

MÜ ist NP-vollständig.

Beweis:

Wir zeigen KNF-SAT $\leq_p$ MÜ.
Sei $F = K_1 \wedge \cdots \wedge K_m$ in KNF, mit Variablen $x_1, \ldots, x_l$.

Wir konstruieren eine Menge M, Teilmengen $T_1, \ldots, T_n$ und eine Zahl k

$$\begin{aligned}
M &:= \{1, \ldots, m, m+1, \ldots, m+l\} \\
T_i &:= \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\} \\
T_{l+i} &:= \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\} \\
n &:= 2l \\
k &:= l
\end{aligned}$$

Satz 6.27

MÜ ist NP-vollständig.

Beweis:

Wir zeigen KNF-SAT $\leq_p$ MÜ.
Sei $F = K_1 \wedge \cdots \wedge K_m$ in KNF, mit Variablen $x_1, \ldots, x_l$.

Wir konstruieren eine Menge M, Teilmengen $T_1, \ldots, T_n$ und eine Zahl k

$$\begin{aligned}
M &:= \{1, \ldots, m, m+1, \ldots, m+l\} \\
T_i &:= \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\} \\
T_{l+i} &:= \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\} \\
n &:= 2l \\
k &:= l
\end{aligned}$$

F ist erfülbar

gdw

M wird durch l der Teilmengen $T_1, \ldots, T_l, T_{l+1}, \ldots, T_{2l}$ überdeckt

□

Das **Minimierungsproblem**

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M

Problem: Finde das kleinste k, so dass M von k der Teilmengen überdeckt wird.

kann auf das Entscheidungsproblem "reduziert" werden:

Das **Minimierungsproblem**

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M

Problem: Finde das kleinste k, so dass M von k der Teilmengen überdeckt wird.

kann auf das Entscheidungsproblem "reduziert" werden:

> Finde kleinstes k durch binäre Suche im Intervall $[1, n]$ mit $O(\log n)$ Aufrufen von MÜ.

Das **Minimierungsproblem**

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M

Problem: Finde das kleinste k, so dass M von k der Teilmengen überdeckt wird.

kann auf das Entscheidungsproblem "reduziert" werden:

> Finde kleinstes k durch binäre Suche im Intervall $[1, n]$ mit $O(\log n)$ Aufrufen von MÜ.

Kann man MÜ in Zeit $O(f(n))$ entscheiden,
dann kann man das kleinste k in Zeit $O(f(n) \cdot \log n)$ berechnen.

$$f(n) \text{ polynomiell} \implies f(n) \cdot \log n \text{ polynomiell}$$
$$f(n) \text{ exponentiell} \implies f(n) \cdot \log n \text{ exponentiell}$$

Die Berechnung einer **Lösung** von

Gegeben: Teilmengen $\vec{T} := T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M, und eine Zahl $k \leq n$.

Problem: Finde eine Überdeckung von M durch k der Teilmengen.

kann auf das Entscheidungsproblem reduziert werden:

Die Berechnung einer **Lösung** von

Gegeben: Teilmengen $\vec{T} := T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M, und eine Zahl $k \leq n$.

Problem: Finde eine Überdeckung von M durch k der Teilmengen.

kann auf das Entscheidungsproblem reduziert werden:

if $(\vec{T}, M, k) \notin$ MÜ **then** output("nicht lösbar")
else
  $\ddot{U} := \emptyset$
  for $i := 1$ **to** n **do**
    if $(\vec{T} - T_i, M, k) \in$ MÜ
    then $\vec{T} := \vec{T} - T_i$
    else $\ddot{U} := \ddot{U} \cup \{T_i\}$

CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

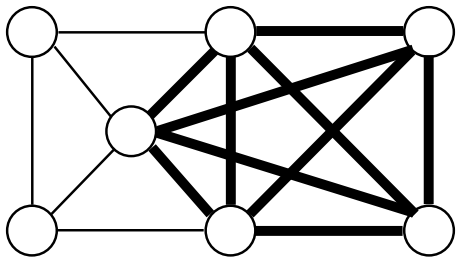
Problem: Besitzt G eine „Clique“ der Größe mindestens k?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:

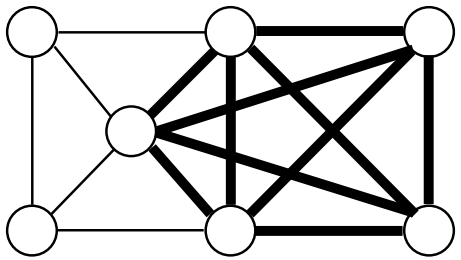


CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:



Anwendung von Clique-Berechnung:

Knoten	=	Prozess
Kante	=	Zwei Prozesse sind parallel ausführbar

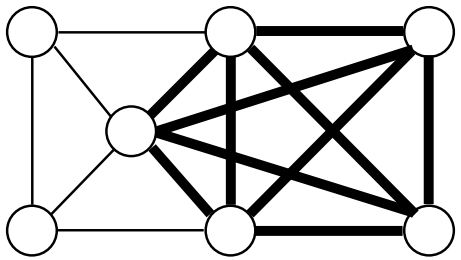
$\Longrightarrow$ Clique ist Gruppe von parallelisierbaren Prozessen

CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:



Anwendung von Clique-Berechnung:

Knoten	=	Prozess
Kante	=	Zwei Prozesse sind parallel ausführbar

$\Longrightarrow$ Clique ist Gruppe von parallelisierbaren Prozessen

Fakt 6.28

CLIQUE $\in$ *NP*

Satz 6.29

CLIQUE ist NP-vollständig.

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$\begin{aligned} V &= \{(1,1),(1,2),(1,3),\ldots,(k,1),(k,2),(k,3)\} \\ E &= \{\{(i,j),(p,q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\} \end{aligned}$$

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$\begin{aligned} V &= \{(1,1),(1,2),(1,3),\ldots,(k,1),(k,2),(k,3)\} \\ E &= \{\{(i,j),(p,q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\} \end{aligned}$$

$$\begin{aligned} & F \text{ ist erfüllbar durch eine Belegung } \sigma \\ \iff & \text{Es gibt in jeder Klausel } i \text{ ein Literal } z_{ij_i} \text{ mit } \sigma(z_{ij_i}) = 1 \\ \iff & \text{Die Literale } z_{1j_1},\ldots,z_{kj_k} \text{ sind paarweise nicht komplementär} \\ \iff & \text{Die Knoten } (1,j_1),\ldots,(k,j_k) \text{ sind paarweise benachbart} \\ \iff & G \text{ hat eine Clique der Größe } k. \end{aligned}$$

$\square$

RUCKSACK

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}^n$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in R} a_i = b$?

RUCKSACK

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}^n$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in R} a_i = b$?

Satz 6.30

RUCKSACK ist NP-vollständig.

RUCKSACK

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}^n$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in R} a_i = b$?

Satz 6.30

RUCKSACK ist NP-vollständig.

Beweis:

3KNF-SAT $\leq_p$ RUCKSACK:

Sei $F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \cdots \wedge (z_{m1} \vee z_{m2} \vee z_{m3})$, wobei

$$z_{ij} \in \{x_1, \ldots, x_n\} \cup \{\neg x_1, \neg x_2, \ldots, \neg x_n\}$$

D.h. $m =$ Anzahl der Klauseln und n Anzahl der Variablen.

Beweis (Forts.):

Wir geben Zahlen v_{i0}, v_{i1} für jede variable x_i, Zahlen k_{j1}, k_{j2} für jede Klausel K_j, und eine Zahl b an.

Beweis (Forts.):

Wir geben Zahlen v_{i0}, v_{i1} für jede variable x_i, Zahlen k_{j1}, k_{j2} für jede Klausel K_j, und eine Zahl b an.

- Alle Zahlen haben genau $m + n$ Stellen im Dezimalsystem.
  Die j-te Stelle, $1 \leq j \leq m$, nennen wir
  "Stelle (der Klausel) K_j".
  Die $m + i$ Stelle, $1 \leq i \leq n$, nennen wir
  "Stelle (der Variable) x_i".

Beweis (Forts.):

Wir geben Zahlen v_{i0}, v_{i1} für jede variable x_i, Zahlen k_{j1}, k_{j2} für jede Klausel K_j, und eine Zahl b an.

- Alle Zahlen haben genau $m + n$ Stellen im Dezimalsystem.
  Die j-te Stelle, $1 \leq j \leq m$, nennen wir
  "Stelle (der Klausel) K_j".
  Die $m + i$ Stelle, $1 \leq i \leq n$, nennen wir
  "Stelle (der Variable) x_i".
- $b = \underbrace{44\ldots444}_{m}\underbrace{11\ldots11}_{n}$

Beweis (Forts.):

Wir geben Zahlen v_{i0}, v_{i1} für jede variable x_i, Zahlen k_{j1}, k_{j2} für jede Klausel K_j, und eine Zahl b an.

- Alle Zahlen haben genau $m + n$ Stellen im Dezimalsystem.
  Die j-te Stelle, $1 \leq j \leq m$, nennen wir
  "Stelle (der Klausel) K_j".
  Die $m + i$ Stelle, $1 \leq i \leq n$, nennen wir
  "Stelle (der Variable) x_i".
- $b = \underbrace{44\ldots444}_{m}\underbrace{11\ldots11}_{n}$
- An der Stelle x_i haben v_{i0}, v_{i1} eine 1, und die Zahlen und k_{j1}, k_{j2} eine 0.
  $\rightarrow$ genau eine von v_{i0}, v_{i1} muss in den Rucksack
  $\rightarrow$ modelliert die Wahl einer Belegung.

Beweis (Forts.):

Beschreibung der Zahlen v_{i0}, v_{i1}, k_{j1}, k_{j2}:

- v_{i0} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die $\neq x_i$ enthalten, sonst 0en.
- v_{i1} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die x_i enthalten, sonst 0en.

Beweis (Forts.):

Beschreibung der Zahlen v_{i0}, v_{i1}, k_{j1}, k_{j2}:

- v_{i0} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die $\neq x_i$ enthalten, sonst 0en.
- v_{i1} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die x_i enthalten, sonst 0en.
- k_{j1} hat eine 1 an der Stell K_j, sonst 0en.
- k_{j2} hat eine 2 an der Stell K_j, sonst 0en.

Beweis (Forts.):

Beschreibung der Zahlen v_{i0}, v_{i1}, k_{j1}, k_{j2}:

- v_{i0} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die $\neq x_i$ enthalten, sonst 0en.
- v_{i1} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die x_i enthalten, sonst 0en.
- k_{j1} hat eine 1 an der Stell K_j, sonst 0en.
- k_{j2} hat eine 2 an der Stell K_j, sonst 0en.

Wenn die Summe einer Untermenge R die Zahl b ergibt, dann erfüllt die Belegung σ mit $\sigma(x_i) = 1$ gdw $v_{i1} \in R$ die Formel F.

Beweis (Forts.):

Beschreibung der Zahlen v_{i0}, v_{i1}, k_{j1}, k_{j2}:

- v_{i0} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die $\neq x_i$ enthalten, sonst 0en.
- v_{i1} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die x_i enthalten, sonst 0en.
- k_{j1} hat eine 1 an der Stell K_j, sonst 0en.
- k_{j2} hat eine 2 an der Stell K_j, sonst 0en.

Wenn die Summe einer Untermenge R die Zahl b ergibt, dann erfüllt die Belegung σ mit $\sigma(x_i) = 1$ gdw $v_{i1} \in R$ die Formel F.

Wenn F erfüllbar ist, dann wähle R so:

- Nehme eine erfüllende Belegung σ von F.
- Für jede Variable x_i: Füge v_{i1} zu R wenn $\sigma(x_i) = 1$, sonst füge v_{i0}.
- Für jede Klausel K_j: Füge k_{j1} oder k_{j2} hinzu, um eine 4 an der Stelle K_j zu gewinnen.

$\square$

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION. Reduktion:

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{n} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

Beispiel: $a_1, \ldots, a_7 = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen $7, 9, 7, 15$, dh $R = \{2, 4, 5, 7\}$

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION. Reduktion:

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{n} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

Beispiel: $a_1, \ldots, a_7 = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen $7, 9, 7, 15$, dh $R = \{2, 4, 5, 7\}$

Es gilt: $M = 57$, $M - b + 1 = 20$, $b + 1 = 39$.

Resultierendes PARTITIONS-Problem: $12, 7, 4, 9, 7, 3, 15, 20, 39$

Lösung: $7 + 9 + 7 + 15 + 20 = 12 + 4 + 3 + 39$. $\square$

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ $a_1, a_2, \ldots a_n$.

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ $a_1, a_2, \ldots a_n$.

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

Satz 6.32

BIN PACKING ist NP-vollständig.

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ $a_1, a_2, \ldots a_n$.

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

Satz 6.32

BIN PACKING ist NP-vollständig.

Beweis: PARTITION $\leq_p$ BIN PACKING. Reduktion:

$$(a_1, \ldots, a_n) \;\mapsto\; (b, k, a_1, \ldots, a_n)$$

wobei $b := \sum_{i=1}^{n} a_i$ und $k := 2$. $\square$

HAMILTONKREIS

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

HAMILTONKREIS

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

Satz 6.33

HAMILTON ist NP-vollständig.

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“ und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge $\leq k$?

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“ und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge $\leq k$?

Satz 6.34

TSP ist NP-vollständig.

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“ und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge $\leq k$?

Satz 6.34

TSP ist NP-vollständig.

Beweis: HAMILTON $\leq_p$ TSP

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“ und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge $\leq k$?

Satz 6.34

TSP ist NP-vollständig.

Beweis: HAMILTON $\leq_p$ TSP

$$(\{1, \ldots, n\}, E) \;\mapsto\; (M, n)$$

wobei

$$M_{ij} \;:=\; \begin{cases} 1 & \text{falls } \{i,j\} \in E \\ 2 & \text{sonst} \end{cases}$$

$\square$

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k.

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k.

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k.

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

Satz 6.35

FÄRBBARKEIT ist NP-vollständig für $k \geq 3$.

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k.

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

Satz 6.35

FÄRBBARKEIT ist NP-vollständig für $k \geq 3$.

Beweis:

3KNF-SAT $\leq_p$ 3FÄRBBARKEIT $\square$

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k.

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

Satz 6.35

FÄRBBARKEIT ist NP-vollständig für $k \geq 3$.

Beweis:

3KNF-SAT $\leq_p$ 3FÄRBBARKEIT $\square$

Satz 6.36

2FÄRBBARKEIT $\in$ P

Die NP-Bibel, der NP-Klassiker:

Michael Garey and David Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

Die NP-Bibel, der NP-Klassiker:

Michael Garey and David Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

> *Despite the 23 years that have passed since its publication, I consider Garey and Johnson the single most important book on my office bookshelf.*

Lance Fortnow, 2002.

Man weiß nicht ob $\mathsf{P} = \mathsf{NP}$.

Man weiß nicht ob $\mathsf{P} = \mathsf{NP}$. Aber man weiß (Ladner 1975)

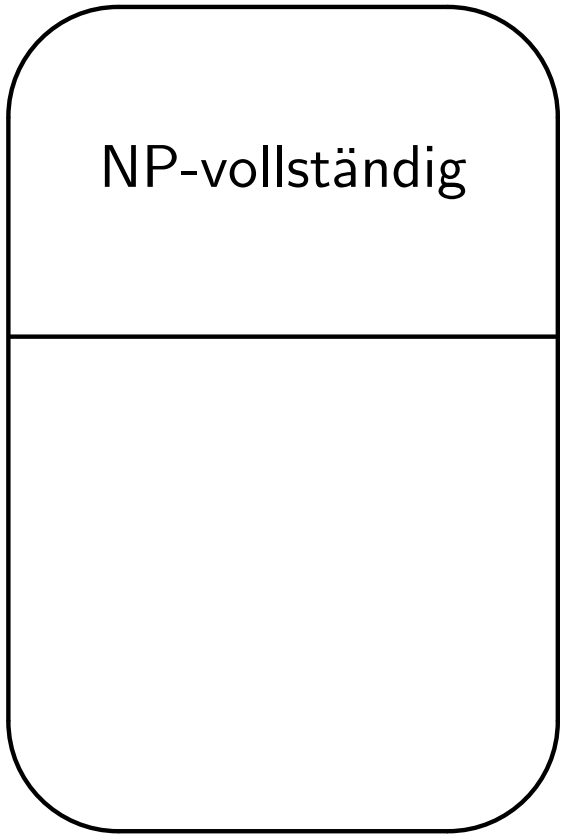
$$\mathsf{P} \neq \mathsf{NP} \quad \Longrightarrow$$

Man weiß nicht ob $\mathsf{P} = \mathsf{NP}$. Aber man weiß (Ladner 1975)

$$\mathsf{P} \neq \mathsf{NP} \quad \Longrightarrow \quad \mathsf{NP} =$$

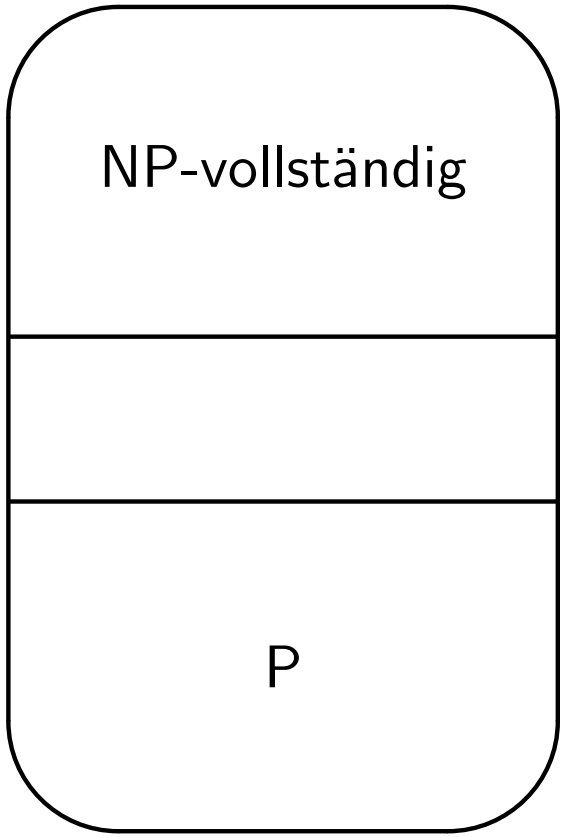
Man weiß nicht ob $\mathsf{P} = \mathsf{NP}$. Aber man weiß (Ladner 1975)

$\mathsf{P} \neq \mathsf{NP} \quad \Longrightarrow \quad \mathsf{NP} =$



Man weiß nicht ob $\mathsf{P} = \mathsf{NP}$. Aber man weiß (Ladner 1975)

$\mathsf{P} \neq \mathsf{NP} \quad \Longrightarrow \quad \mathsf{NP} =$



6.5 Efficiency beyond efficiency

NP-hard problems

- not believed to be "efficiently" solvable, i.e., in polynomial time
- *NP-complete*: many combinatorial/graph problems, satisfiability of a propositional-logic formula (SAT)
- even harder: many problems in AI, verification, ...

Today: What to do with NP-complete problems?

6.5 Efficiency beyond efficiency

NP-hard problems

- not believed to be "efficiently" solvable, i.e., in polynomial time
- *NP-complete*: many combinatorial/graph problems, satisfiability of a propositional-logic formula (SAT)
- even harder: many problems in AI, verification, ...

Today: What to do with NP-complete problems?

- more computational power?

6.5 Efficiency beyond efficiency

NP-hard problems

- not believed to be "efficiently" solvable, i.e., in polynomial time
- *NP-complete*: many combinatorial/graph problems, satisfiability of a propositional-logic formula (SAT)
- even harder: many problems in AI, verification, ...

Today: What to do with NP-complete problems?

- more computational power?
- encode into SAT

6.5 Efficiency beyond efficiency

NP-hard problems

- not believed to be "efficiently" solvable, i.e., in polynomial time
- *NP-complete*: many combinatorial/graph problems, satisfiability of a propositional-logic formula (SAT)
- even harder: many problems in AI, verification, ...

Today: What to do with NP-complete problems?

- more computational power?
- encode into SAT
- approximation algorithms

Travelling Salesman Problem

Definition 6.37 (TSP)

Given a *complete*, weighted, undirected graph $G = (V, E)$ with non-negative weights $c\colon V \to \mathbb{N}$, find a cycle that visits exactly all nodes and does so with *minimal length*.

Travelling Salesman Problem

Definition 6.37 (TSP)

Given a *complete*, weighted, undirected graph $G = (V, E)$ with non-negative weights $c\colon V \to \mathbb{N}$, find a cycle that visits exactly all nodes and does so with *minimal length*.

Properties

- We can assume *triangle inequality*:

$$\forall u, v, w \in V. c(u, v) \leq c(u, w) + c(w, v)$$

- NP-complete
- We show a 2-approximation

Travelling Salesman Problem

Definition 6.37 (TSP)

Given a *complete*, weighted, undirected graph $G = (V, E)$ with non-negative weights $c\colon V \to \mathbb{N}$, find a cycle that visits exactly all nodes and does so with *minimal length*.

Properties

- We can assume *triangle inequality*:

$$\forall u, v, w \in V. c(u, v) \leq c(u, w) + c(w, v)$$

- NP-complete
- We show a 2-approximation
- There is a 1.5-approximation

Travelling Salesman Problem

Definition 6.37 (TSP)

Given a *complete*, weighted, undirected graph $G = (V, E)$ with non-negative weights $c\colon V \to \mathbb{N}$, find a cycle that visits exactly all nodes and does so with *minimal length*.

Properties

- We can assume *triangle inequality*:

$$\forall u, v, w \in V. c(u, v) \leq c(u, w) + c(w, v)$$

- NP-complete
- We show a 2-approximation
- There is a 1.5-approximation
- There is no 123/122-approximation (since 2015)

2-Approximation Algorithm for TSP

Algorithm

1. T := a minimum spanning tree
2. cycle := traverse along depth-first search of T, jumping over visited nodes

2-Approximation Algorithm for TSP

Algorithm

1. T := a minimum spanning tree
2. cycle := traverse along depth-first search of T, jumping over visited nodes

Algorithm is

- *polynomial*
- *2-approximation*
  - $c(T) \leq$ minimal cycle
  - traversal costs $2 \cdot c(T)$ since jumping over costs at most the sum of traversed edges

Knapsack

Definition 6.38 (Knapsack)

Given weight W of knapsack and *weights* and *values* of n items: $w_1, \ldots, w_m, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal (under the previous constraint).

Knapsack

Definition 6.38 (Knapsack)

Given weight W of knapsack and *weights* and *values* of n items: $w_1, \ldots, w_m, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal (under the previous constraint).

Greedy Algorithm

- take items in the order $v_1/w_1 \geq v_2/w_2 \cdots \geq v_n/w_n$

Knapsack

Definition 6.38 (Knapsack)

Given weight W of knapsack and *weights* and *values* of n items: $w_1, \ldots, w_m, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal (under the previous constraint).

Greedy Algorithm

- take items in the order $v_1/w_1 \geq v_2/w_2 \cdots \geq v_n/w_n$

Properties

- optimal for "fractional" knapsack problem
- for $v_1 = 1.001, w_1 = 1, v_2 = W, w_2 = W$ no better than a W-approximation.

2-Approximation of Knapsack

Modified Greedy Algorithm (ModGreedy):

- $S_1 :=$ solution by Greedy
- $S_2 :=$ item with the largest value
- Return whichever of S_1, S_2 that has more value

Lemma 6.39

ModGreedy is a 2-approximation.

2-Approximation of Knapsack

Modified Greedy Algorithm (ModGreedy):

- S_1 := solution by Greedy
- S_2 := item with the largest value
- Return whichever of S_1, S_2 that has more value

Lemma 6.39

ModGreedy is a 2-approximation.

Proof.

- If Greedy takes items $1, 2, \ldots, k-1$, then $\sum_{i=1}^{k} v_i \geq OPT_{frac} \geq OPT$: kth item might not be taken in full + the optimal integral solution is not better than the optimal fractional solution
- $(v_1 + \cdots + v_{k-1}) + v_k \geq OPT$
- one of the two is $\geq OPT/2$
- $v(S_1) = \sum_{i=1}^{k-1} v_i$, and $v(S_2) = v_{\max} \geq v_k$

$\square$

PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible
- Idea: brute-force a part of the solution and then use Greedy Algorithm to finish up the rest

Algorithm, k fixed constant

- for all possible subsets of objects that have up to k objects:
- use the greedy algorithm to fill up the rest of the knapsack
- return the most profitable subset

PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible
- Idea: brute-force a part of the solution and then use Greedy Algorithm to finish up the rest

Algorithm, k fixed constant

- for all possible subsets of objects that have up to k objects:
- use the greedy algorithm to fill up the rest of the knapsack
- return the most profitable subset

Properties

- runtime $\mathcal{O}(kn^k)$ subsets, filling up in $\mathcal{O}(n)$
- thus total running time $\mathcal{O}(kn^{k+1})$
- $(1 + \frac{1}{k})-$approximation

Kurt Gödel.
Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik, 1931.

Kurt Gödel.
Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik, 1931.

Kurt Gödel
1906 (Brünn) –
1978 (Princeton)

6.6 Die Unvollständigkeit der Arithmetik

6.6 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen: $V \qquad x\ y\ z\ \ldots$

6.6 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen:	V	$x\ y\ z\ \ldots$
Zahlen:	N	$0\ 1\ 2\ \ldots$

6.6 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen:	V	$x\ y\ z\ \ldots$
Zahlen:	N	$0\ 1\ 2\ \ldots$
Terme:	T	$V\ N\ (T+T)\ (T*T)$

6.6 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen:	V	$x\ y\ z\ \ldots$
Zahlen:	N	$0\ 1\ 2\ \ldots$
Terme:	T	$V\ \ N\ \ (T+T)\ (T*T)$
Formeln:	F	$(T=T)\ \neg F\ (F \wedge F)\ (F \vee F)$
		$\exists V.\, F$

6.6 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen:	V	$x\ y\ z\ \ldots$
Zahlen:	N	$0\ 1\ 2\ \ldots$
Terme:	T	$V\ N\ (T+T)\ (T*T)$
Formeln:	F	$(T=T)\ \neg F\ (F \wedge F)\ (F \vee F)$
		$\exists V.\, F$

Wir betrachten $\forall x.\, F$ als Abk. für $\neg\exists\ x.\, \neg F$.

6.6 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen:	V	$x\ y\ z\ \ldots$
Zahlen:	N	$0\ 1\ 2\ \ldots$
Terme:	T	$V\ N\ (T+T)\ (T*T)$
Formeln:	F	$(T=T)\ \neg F\ (F \wedge F)\ (F \vee F)$
		$\exists V.\, F$

Wir betrachten $\forall x.\, F$ als Abk. für $\neg\exists\ x.\, \neg F$.

Definition 6.40

Ein Vorkommen einer Variablen x in einer Formel F ist gebunden gdw das Vorkommen in einer Teilformel der Form $\exists x.\, F'$ oder $\forall x.\, F'$ in der Teilformel F' liegt.

6.6 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen:	V	$x\ y\ z\ \ldots$
Zahlen:	N	$0\ 1\ 2\ \ldots$
Terme:	T	$V\ N\ (T+T)\ (T*T)$
Formeln:	F	$(T=T)\ \neg F\ (F \wedge F)\ (F \vee F)$
		$\exists V.\, F$

Wir betrachten $\forall x.\, F$ als Abk. für $\neg\exists\ x.\, \neg F$.

Definition 6.40

Ein Vorkommen einer Variablen x in einer Formel F ist gebunden gdw das Vorkommen in einer Teilformel der Form $\exists x.\, F'$ oder $\forall x.\, F'$ in der Teilformel F' liegt.
Ein Vorkommen ist frei gdw es nicht gebunden ist.

Notation: $F(x_1, \ldots, x_k)$ bezeichnet eine Formel F, in der höchstens $x_1, \ldots, x_k$ frei vorkommen.

Notation: $F(x_1, \ldots, x_k)$ bezeichnet eine Formel F, in der höchstens $x_1, \ldots, x_k$ frei vorkommen.

Sind $n_1, \ldots, n_k \in \mathbb{N}$ so ist $F(n_1, \ldots, n_k)$ das Ergebnis der Substitution von $n_1, \ldots, n_k$ für die freien Vorkommen von $x_1, \ldots, x_k$.

Notation: $F(x_1, \ldots, x_k)$ bezeichnet eine Formel F, in der höchstens $x_1, \ldots, x_k$ frei vorkommen.

Sind $n_1, \ldots, n_k \in \mathbb{N}$ so ist $F(n_1, \ldots, n_k)$ das Ergebnis der Substitution von $n_1, \ldots, n_k$ für die freien Vorkommen von $x_1, \ldots, x_k$.

Beispiel 6.41

$$F(x, y) \quad = \quad (x = y \ \wedge \ \exists x.\, x = y)$$

Notation: $F(x_1, \ldots, x_k)$ bezeichnet eine Formel F, in der höchstens $x_1, \ldots, x_k$ frei vorkommen.

Sind $n_1, \ldots, n_k \in \mathbb{N}$ so ist $F(n_1, \ldots, n_k)$ das Ergebnis der Substitution von $n_1, \ldots, n_k$ für die freien Vorkommen von $x_1, \ldots, x_k$.

Beispiel 6.41

$$\begin{aligned} F(x, y) &= (x = y \ \wedge \ \exists x.\, x = y) \\ F(5, 7) &= (5 = 7 \ \wedge \ \exists x.\, x = 7) \end{aligned}$$

Notation: $F(x_1, \ldots, x_k)$ bezeichnet eine Formel F, in der höchstens $x_1, \ldots, x_k$ frei vorkommen.

Sind $n_1, \ldots, n_k \in \mathbb{N}$ so ist $F(n_1, \ldots, n_k)$ das Ergebnis der Substitution von $n_1, \ldots, n_k$ für die freien Vorkommen von $x_1, \ldots, x_k$.

Beispiel 6.41

$$\begin{aligned} F(x, y) &= (x = y \ \wedge \ \exists x.\, x = y) \\ F(5, 7) &= (5 = 7 \ \wedge \ \exists x.\, x = 7) \end{aligned}$$

Ein Satz ist eine Formel ohne freie Variablen.

Notation: $F(x_1, \ldots, x_k)$ bezeichnet eine Formel F, in der höchstens $x_1, \ldots, x_k$ frei vorkommen.

Sind $n_1, \ldots, n_k \in \mathbb{N}$ so ist $F(n_1, \ldots, n_k)$ das Ergebnis der Substitution von $n_1, \ldots, n_k$ für die freien Vorkommen von $x_1, \ldots, x_k$.

Beispiel 6.41

$$\begin{aligned} F(x, y) &= (x = y \ \wedge \ \exists x.\, x = y) \\ F(5, 7) &= (5 = 7 \ \wedge \ \exists x.\, x = 7) \end{aligned}$$

Ein Satz ist eine Formel ohne freie Variablen.

Beispiel 6.42

$$\exists x.\ \exists y.\, x = y \qquad \exists y.\ \exists x.\, x = y$$

Notation: $F(x_1, \ldots, x_k)$ bezeichnet eine Formel F, in der höchstens $x_1, \ldots, x_k$ frei vorkommen.

Sind $n_1, \ldots, n_k \in \mathbb{N}$ so ist $F(n_1, \ldots, n_k)$ das Ergebnis der Substitution von $n_1, \ldots, n_k$ für die freien Vorkommen von $x_1, \ldots, x_k$.

Beispiel 6.41

$$\begin{aligned} F(x,y) &= (x = y \ \wedge \ \exists x.\, x = y) \\ F(5,7) &= (5 = 7 \ \wedge \ \exists x.\, x = 7) \end{aligned}$$

Ein Satz ist eine Formel ohne freie Variablen.

Beispiel 6.42

$$\exists x.\ \exists y.\, x = y \qquad \exists y.\ \exists x.\, x = y$$

Mit S bezeichnen wir die Menge der arithmetischen Sätze.

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 6.43

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 6.43

$(t_1 = t_2) \in W$ gdw t_1 und t_2 den selben Wert haben.

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 6.43

$$(t_1 = t_2) \in W \quad \text{gdw} \quad t_1 \text{ und } t_2 \text{ den selben Wert haben.}$$

$$\neg F \in W \quad \text{gdw} \quad F \notin W$$

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 6.43

$$
\begin{aligned}
(t_1 = t_2) \in W \quad & \text{gdw} \quad t_1 \text{ und } t_2 \text{ den selben Wert haben.} \\
\neg F \in W \quad & \text{gdw} \quad F \notin W \\
(F \wedge G) \in W \quad & \text{gdw} \quad F \in W \text{ und } G \in W
\end{aligned}
$$

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 6.43

$(t_1 = t_2) \in W$	gdw	t_1 und t_2 den selben Wert haben.
$\neg F \in W$	gdw	$F \notin W$
$(F \wedge G) \in W$	gdw	$F \in W$ und $G \in W$
$(F \vee G) \in W$	gdw	$F \in W$ oder $G \in W$

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 6.43

$(t_1 = t_2) \in W$	gdw	t_1 und t_2 den selben Wert haben.
$\neg F \in W$	gdw	$F \notin W$
$(F \wedge G) \in W$	gdw	$F \in W$ und $G \in W$
$(F \vee G) \in W$	gdw	$F \in W$ oder $G \in W$
$\exists x.\, F(x) \in W$	gdw	es $n \in \mathbb{N}$ gibt, so dass $F(n) \in W$

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 6.43

$(t_1 = t_2) \in W$	gdw	t_1 und t_2 den selben Wert haben.
$\neg F \in W$	gdw	$F \notin W$
$(F \wedge G) \in W$	gdw	$F \in W$ und $G \in W$
$(F \vee G) \in W$	gdw	$F \in W$ oder $G \in W$
$\exists x.\, F(x) \in W$	gdw	es $n \in \mathbb{N}$ gibt, so dass $F(n) \in W$

Fakt 6.44

Für jeden Satz F gilt entweder $F \in W$ oder $\neg F \in W$

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 6.43

$(t_1 = t_2) \in W$	gdw	t_1 und t_2 den selben Wert haben.
$\neg F \in W$	gdw	$F \notin W$
$(F \wedge G) \in W$	gdw	$F \in W$ und $G \in W$
$(F \vee G) \in W$	gdw	$F \in W$ oder $G \in W$
$\exists x.\, F(x) \in W$	gdw	es $n \in \mathbb{N}$ gibt, so dass $F(n) \in W$

Fakt 6.44

Für jeden Satz F gilt entweder $F \in W$ oder $\neg F \in W$

NB Ob eine Formel mit freien Variablen wahr ist, kann vom Wert der freien Variablen abhängen:

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 6.43

$(t_1 = t_2) \in W$	gdw	t_1 und t_2 den selben Wert haben.
$\neg F \in W$	gdw	$F \notin W$
$(F \wedge G) \in W$	gdw	$F \in W$ und $G \in W$
$(F \vee G) \in W$	gdw	$F \in W$ oder $G \in W$
$\exists x.\, F(x) \in W$	gdw	es $n \in \mathbb{N}$ gibt, so dass $F(n) \in W$

Fakt 6.44

Für jeden Satz F gilt entweder $F \in W$ oder $\neg F \in W$

NB Ob eine Formel mit freien Variablen wahr ist, kann vom Wert der freien Variablen abhängen:

$$\exists x.\, x + x = y$$

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 6.43

$(t_1 = t_2) \in W$	gdw	t_1 und t_2 den selben Wert haben.
$\neg F \in W$	gdw	$F \notin W$
$(F \wedge G) \in W$	gdw	$F \in W$ und $G \in W$
$(F \vee G) \in W$	gdw	$F \in W$ oder $G \in W$
$\exists x.\, F(x) \in W$	gdw	es $n \in \mathbb{N}$ gibt, so dass $F(n) \in W$

Fakt 6.44

Für jeden Satz F gilt entweder $F \in W$ oder $\neg F \in W$

NB Ob eine Formel mit freien Variablen wahr ist, kann vom Wert der freien Variablen abhängen:

$$\exists x.\, x + x = y$$

Daher haben wir Wahrheit nur für Sätze definiert.

Definition 6.45

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist arithmetisch repräsentierbar gdw es eine Formel $F(x_1, \ldots, x_k, y)$ gibt, so dass für alle $n_1, \ldots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \ldots, n_k) = m \quad \text{gdw} \quad F(n_1, \ldots, n_k, m) \in W$$

Definition 6.45

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist arithmetisch repräsentierbar gdw es eine Formel $F(x_1, \ldots, x_k, y)$ gibt, so dass für alle $n_1, \ldots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \ldots, n_k) = m \quad \text{gdw} \quad F(n_1, \ldots, n_k, m) \in W$$

Satz 6.46

Jede WHILE-berechenbare Funktion ist arithmetisch repräsentierbar.

Satz 6.47

W ist nicht entscheidbar.

Korollar 6.48

W ist nicht semi-entscheidbar.

Wir kodieren Beweise als Zahlen.

Definition 6.49

Ein Beweissystem für die Arithmetik ist ein entscheidbares Prädikat

$$Bew : \mathbb{N} \times S \;\rightarrow\; \{0, 1\}$$

wobei wir $Bew(b, F)$ lesen als „b ist Beweis für Formel F“.
Ein Beweissystem Bew ist korrekt gdw

$$Bew(b, F) \implies F \in W.$$

Ein Beweissystem Bew ist vollständig gdw

$$F \in W \implies \text{ es gibt } b \text{ mit } Bew(b, F).$$

Satz 6.50 (Gödel)

Es gibt kein korrektes und vollständiges Beweissystem für die Arithmetik.

Beweis:

Denn mit jedem korrekten und vollständigen Beweissystem kann man $\chi'_W(F)$ programmieren:

$b := 0$
while $Bew(b, F) = 0$ **do** $b := b + 1$
output(1)

$\square$

Gödel's 1st incompleteness theorem

Theorem 6.51

There are true arithmetical formulae unprovable in PA (or other consistent formal systems).

Gödel's proof (sketch)

- it is possible to construct a PA formula ρ such that

$$PA \quad \vdash \quad \rho \Leftrightarrow \neg\texttt{Provable}([\rho])$$

  i.e. "ρ says "I'm not provable"" is provable in PA
- by consistency of PA this is true in arithmetics
- if $\neg\rho$ then $\texttt{Provable}([\rho])$, a contradiction
  if ρ then $\neg\texttt{Provable}([\rho])$ hence $PA \nvdash \rho$ $\square$

Gödel's 1st incompleteness theorem

Recall:

- H_0 is r.e., but not recursive
- $\overline{H_0}$ is not r.e.

Alternative proof:

- `Provable` is r.e. (for PA and similar)
- `Provable` $\subseteq$ `Valid` by consistency
- we prove `Valid` is not r.e., hence $\subsetneq$
  - $\overline{H_0} \leq$ `Valid`:
    construct a program transforming $n \in \mathbb{N}$ into a formula φ:

    $$\varphi \in \texttt{Valid} \quad \text{iff} \quad n \in \overline{H_0}$$

    it computes the formula "M_n does not accept"
  - computation is a sequence of configurations (numbers)
  - one can encode that a configuration c follows a given configuration d
  - every finite sequence can be encoded by a formula β

Gödel's 1st incompleteness theorem

Let $\beta(a, b, i, x)$ be true iff $x = a \bmod (1 + b(1 + i))$

- expressible in simple arithmetics:

  $$a \geq 0 \wedge b \geq 0 \wedge \exists k \big(k \geq 0 \wedge k * c \leq a \wedge (k+1) * c > a \wedge x = a - (k * b)\big)$$

  where c is a shortcut for $(1 + b * (1 + i))$
- for every a, b the predicate β induces a unique sequence, where the ith element is $a \bmod (1 + b(1 + i))$
- every finite sequence can be encoded by β for some a, b:

Theorem 6.52

For every $n_1, \ldots, n_k$ there are $a, b \in \mathbb{N}$ such that

$$\beta(a, b, i, x) \text{ iff } x = n_i$$

Gödel's 1st incompleteness theorem

$\beta(a, b, i, x)$ iff $x = a \bmod (1 + b(1 + i))$

Theorem 6.53

For every $n_1, \ldots, n_k$ there are $a, b \in \mathbb{N}$ such that

$$\beta(a, b, i, x) \text{ iff } x = n_i$$

Proof.

- $b := (\max\{k, n_1, \ldots, n_k\})!$
- $p_i := 1 + b(1 + i)$ is $\geq n_i$ and mutually incommensurable
- $c_i := \prod_{j \neq i} p_j$
- $\exists! \quad 0 \leq d_i \leq p_i : c_i \cdot d_i \bmod p_i = 1$
- $a := \sum_{i=1}^{k} c_i \cdot d_i \cdot n_i$
- hence $n_i = a \bmod p_i$

$\square$

6.7 Die Entscheidbarkeit der Presburger Arithmetik

6.7 Die Entscheidbarkeit der Presburger Arithmetik

Syntax der Presburger Arithmetik:

$$
\begin{array}{rcll}
\text{Variablen:} & V & \rightarrow & x \mid y \mid z \mid \ldots \\
\text{Zahlen:} & N & \rightarrow & 0 \mid 1 \mid 2 \mid \ldots \\
\text{Terme:} & T & \rightarrow & V \mid N \mid T + T \\
\text{Formeln:} & F & \rightarrow & (T = T) \mid \neg F \mid (F \wedge F) \mid (F \vee F) \\
 & & \mid & \exists V.\, F
\end{array}
$$

6.7 Die Entscheidbarkeit der Presburger Arithmetik

Syntax der Presburger Arithmetik:

$$
\begin{array}{rcll}
\text{Variablen:} & V & \rightarrow & x \mid y \mid z \mid \ldots \\
\text{Zahlen:} & N & \rightarrow & 0 \mid 1 \mid 2 \mid \ldots \\
\text{Terme:} & T & \rightarrow & V \mid N \mid T + T \\
\text{Formeln:} & F & \rightarrow & (T = T) \mid \neg F \mid (F \wedge F) \mid (F \vee F) \\
 & & \mid & \exists V.\, F
\end{array}
$$

Wir betrachten $\forall x.\, F$ als Abk. für $\neg \exists x.\, \neg F$.

6.7 Die Entscheidbarkeit der Presburger Arithmetik

Syntax der Presburger Arithmetik:

$$
\begin{array}{rcll}
\text{Variablen:} & V & \rightarrow & x \mid y \mid z \mid \ldots \\
\text{Zahlen:} & N & \rightarrow & 0 \mid 1 \mid 2 \mid \ldots \\
\text{Terme:} & T & \rightarrow & V \mid N \mid T + T \\
\text{Formeln:} & F & \rightarrow & (T = T) \mid \neg F \mid (F \wedge F) \mid (F \vee F) \\
 & & \mid & \exists V.\, F
\end{array}
$$

Wir betrachten $\forall x.\, F$ als Abk. für $\neg \exists x.\, \neg F$.

Satz 6.54

Für Sätze der Presburger Arithmetik ist entscheidbar, ob sie wahr sind.

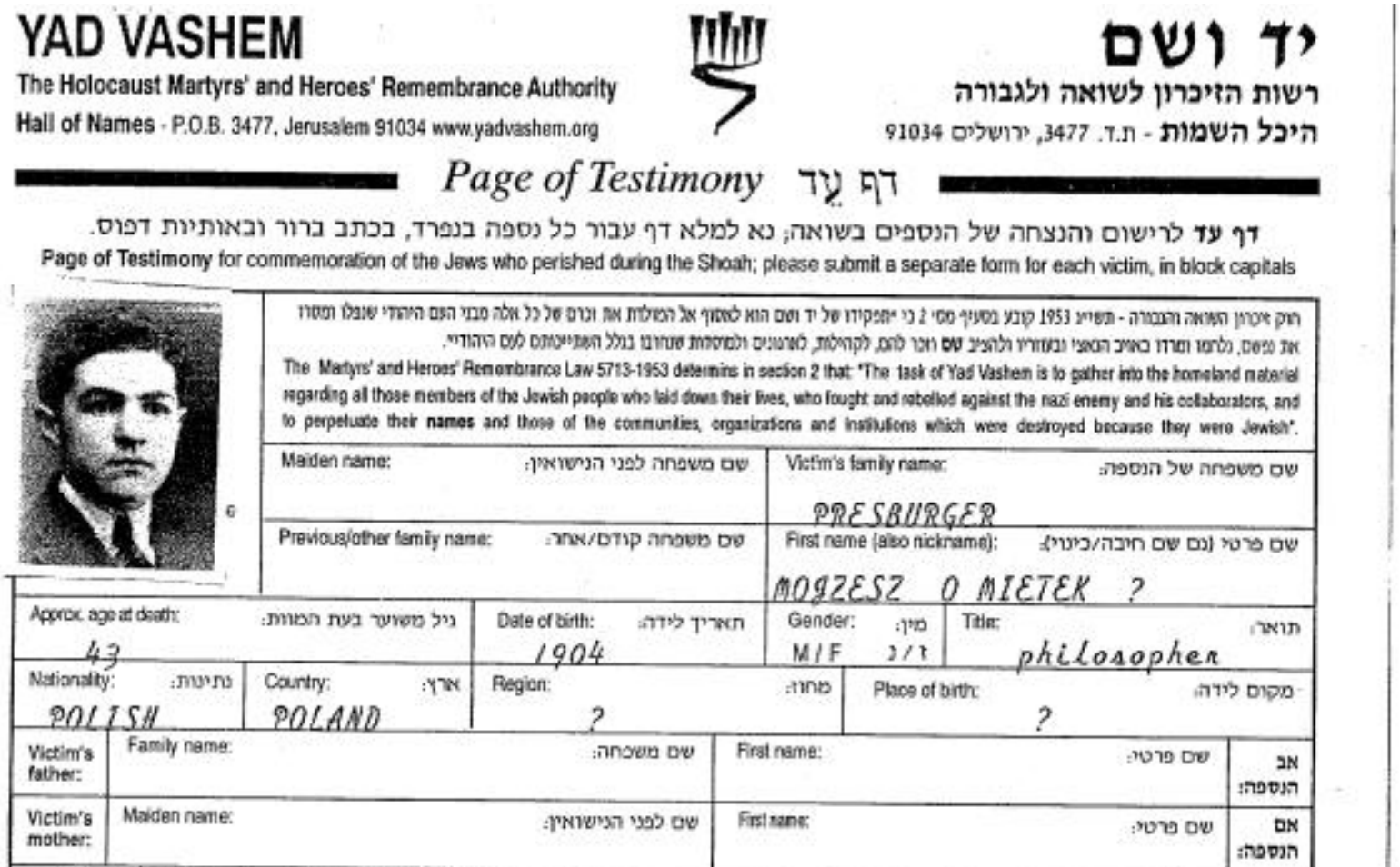
Mojzesz Presburger.
Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. 1929.

Mojzesz Presburger.
Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. 1929.

Mojzesz Presburger 1904 – 1943

Mojzesz Presburger.
Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. 1929.

Mojzesz Presburger 1904 – 1943



YAD VASHEM
The Holocaust Martyrs' and Heroes' Remembrance Authority
Hall of Names - P.O.B. 3477, Jerusalem 91034 www.yadvashem.org

יד ושם
רשות הזיכרון לשואה ולגבורה
היכל השמות - ת.ד. 3477, ירושלים 91034

Page of Testimony דף עֵד

דף עד לרישום והנצחה של הנספים בשואה; נא למלא דף עבור כל נספה בנפרד, בכתב ברור ובאותיות דפוס.
Page of Testimony for commemoration of the Jews who perished during the Shoah; please submit a separate form for each victim, in block capitals

חוק איכרון השואה והגבורה - תשיג 1953 קובע בסעיף מס' 2 כי יתפקידו של יד ושם הוא לאסוף אל המולדת את זכרם של כל אלה מבני העם היהודי שנפלו ומסרו את נפשם, נלחמו ומרדו באויב הנאצי ובעזריו ולהנציח שם וזכר להם, לקהילות, לארגונים ולמוסדות שנהרסו בגלל השתייכותם לעם היהודי.
The Martyrs' and Heroes' Remembrance Law 5713-1953 determins in section 2 that: "The task of Yad Vashem is to gather into the homeland material regarding all those members of the Jewish people who laid down their lives, who fought and rebelled against the nazi enemy and his collaborators, and to perpetuate their **names** and those of the communities, organizations and institutions which were destroyed because they were Jewish".

Maiden name: שם משפחה לפני הנישואין:		Victim's family name: שם משפחה של הנספה:	PRESBURGER
Previous/other family name: שם משפחה קודם/אחר:		First name (also nickname): שם פרטי (גם שם חיבה/כינוי):	MOJZESZ O MIETEK ?
Approx. age at death: גיל משוער בעת המוות:	43	Date of birth: תאריך לידה:	1904
Gender: מין:	M/F ז/נ	Title: תואר:	philosopher
Nationality: נתינות:	POLISH	Country: ארץ:	POLAND
Region: מחוז:	?	Place of birth: מקום לידה:	?
Victim's father:	Family name: שם משפחה:	First name: שם פרטי:	אב הנספה:
Victim's mother:	Maiden name: שם לפני הנישואין:	First name: שם פרטי:	אם הנספה:

Presburger Arithmetik $=$ normale Arithmetik
ohne Multiplikation

Arithmetik : hochgradig unentscheidbar :-(
sogar sogar unvollständig :-((

$\Longrightarrow$ Hilberts 10tes Problem
$\Longrightarrow$ Gödels Theorem

Vereinfachte Presburger Formeln:

$$
\begin{array}{lcl}
\phi & ::= & x + y = z \quad | \quad x = n \quad | \\
 & & \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \\
 & & \exists\, x.\ \ \phi
\end{array}
$$

Vereinfachte Presburger Formeln:

$$\begin{array}{lcl} \phi & ::= & x + y = z \quad | \quad x = n \quad | \\ & & \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \\ & & \exists\, x.\ \ \phi \end{array}$$

Bemerkungen

- Durch sukzessive Addition kann man Multiplikation mit Konstanten simulieren.
  (Wie?)

Vereinfachte Presburger Formeln:

$$\begin{array}{lcl} \phi & ::= & x + y = z \quad | \quad x = n \quad | \\ & & \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \\ & & \exists\, x.\ \phi \end{array}$$

Bemerkungen

- Durch sukzessive Addition kann man Multiplikation mit Konstanten simulieren.
  (Wie?)

- Das Rucksackproblem lässt sich in PA ausdrücken.
  (Wie?)

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel: **PSAT**

Finde Werte in $\mathbb{N}$ für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken. (Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist regulär !!

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist regulär !!

$$\begin{array}{lcll} \phi_1 \wedge \phi_2 & \Longrightarrow & \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) & \text{(Durchschnitt)} \\ \neg\phi & \Longrightarrow & \overline{\mathcal{L}(\phi)} & \text{(Komplement)} \\ \exists\, x : \quad \phi & \Longrightarrow & \pi_x(\mathcal{L}(\phi)) & \text{(Projektion)} \end{array}$$

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist regulär !!

$$\begin{array}{lcll} \phi_1 \wedge \phi_2 & \Longrightarrow & \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) & \text{(Durchschnitt)} \\ \neg\phi & \Longrightarrow & \overline{\mathcal{L}(\phi)} & \text{(Komplement)} \\ \exists\, x : \quad \phi & \Longrightarrow & \pi_x(\mathcal{L}(\phi)) & \text{(Projektion)} \end{array}$$

Achtung:

- Ein akzeptiertes Tupel kann immer durch führende Nullen verlängert werden !
- bleibt unter Vereinigung, Durchschnitt und Komplement erhalten !!

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist regulär !!

$$\begin{array}{lcll} \phi_1 \wedge \phi_2 & \Longrightarrow & \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) & \text{(Durchschnitt)} \\ \neg\phi & \Longrightarrow & \overline{\mathcal{L}(\phi)} & \text{(Komplement)} \\ \exists\, x : \quad \phi & \Longrightarrow & \pi_x(\mathcal{L}(\phi)) & \text{(Projektion)} \end{array}$$

Achtung:

- Ein akzeptiertes Tupel kann immer durch führende Nullen verlängert werden !
- bleibt unter Vereinigung, Durchschnitt und Komplement erhalten !!
- Wie sieht das mit der Projektion aus ?

Weg-Projizierung der x-Komponente:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Weg-Projizierung der x-Komponente:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0

Weg-Projizierung der x-Komponente:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0

- Nun werden möglicherweise Tupel von Zahlen erst nach Anhängen von geeignet vielen führenden Nullen akzeptiert !

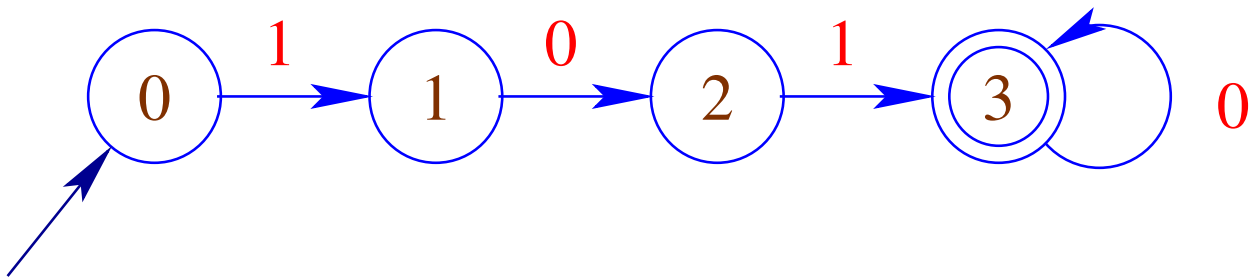
Weg-Projizierung der x-Komponente:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0

- Nun werden möglicherweise Tupel von Zahlen erst nach Anhängen von geeignet vielen führenden Nullen akzeptiert !
- Der Automat muss so komplettiert werden, dass q bereits akzeptierend ist, wenn von q aus mit Nullen ein akzeptierender Zustand erreicht werden kann.

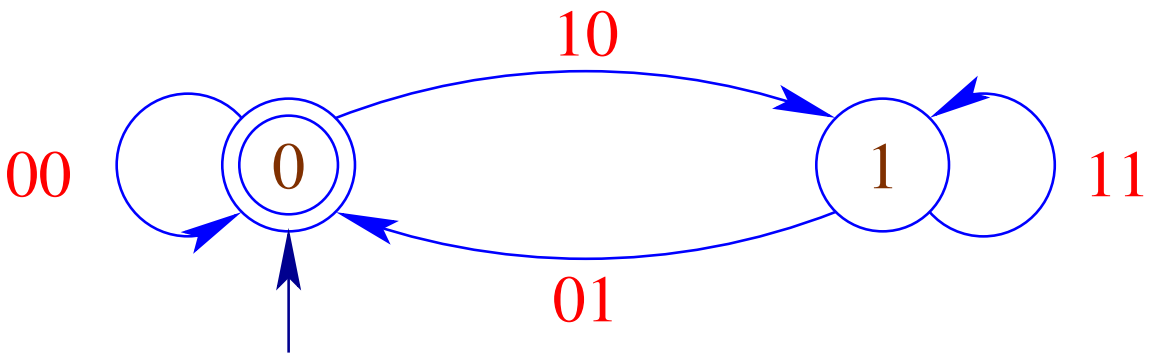
Automaten für Basis-Prädikate:

$$x = 5$$



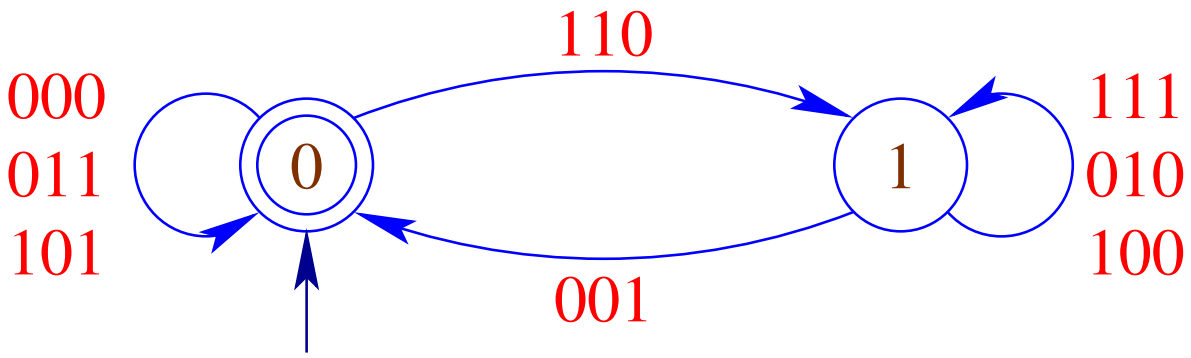
Automaten für Basis-Prädikate:

$$x+x = y$$



Automaten für Basis-Prädikate:

$$x+y = z$$



Ergebnisse:

Ferrante, Rackoff,1973 : $\mathrm{PSAT} \leq \mathrm{DSPACE}(2^{2^{c \cdot n}})$

Ergebnisse:

Ferrante, Rackoff,1973 : $\mathrm{PSAT} \leq \mathrm{DSPACE}(2^{2^{c \cdot n}})$

Fischer, Rabin,1974 : $\mathrm{PSAT} \geq \mathrm{NTIME}(2^{2^{c \cdot n}})$

Informatik im Jahr 2100: Eine Vorhersage

Informatik im Jahr 2100: Eine Vorhersage

Java 7 $\rightsquigarrow$

Informatik im Jahr 2100: Eine Vorhersage

Java 7 $\rightsquigarrow$ **Taiwan (Red Dragon Edition)**

Informatik im Jahr 2100: Eine Vorhersage

Java 7	$\rightsquigarrow$	**Taiwan (Red Dragon Edition)**
Python	$\rightsquigarrow$	

Informatik im Jahr 2100: Eine Vorhersage

Java 7	$\rightsquigarrow$	**Taiwan (Red Dragon Edition)**
Python	$\rightsquigarrow$	**Gnu-Boa++# 18.3**

Informatik im Jahr 2100: Eine Vorhersage

Java 7	$\rightsquigarrow$	**Taiwan (Red Dragon Edition)**
Python	$\rightsquigarrow$	**Gnu-Boa++# 18.3**
XML	$\rightsquigarrow$	

Informatik im Jahr 2100: Eine Vorhersage

Java 7	$\rightsquigarrow$	**Taiwan (Red Dragon Edition)**
Python	$\rightsquigarrow$	**Gnu-Boa++# 18.3**
XML	$\rightsquigarrow$	**Medea**

Informatik im Jahr 2100: Eine Vorhersage

Java 7	$\rightsquigarrow$	**Taiwan (Red Dragon Edition)**
Python	$\rightsquigarrow$	**Gnu-Boa++# 18.3**
XML	$\rightsquigarrow$	**Medea**
Reguläre Sprachen	$\rightsquigarrow$	

Informatik im Jahr 2100: Eine Vorhersage

Java 7	$\rightsquigarrow$	**Taiwan (Red Dragon Edition)**
Python	$\rightsquigarrow$	**Gnu-Boa++# 18.3**
XML	$\rightsquigarrow$	**Medea**
Reguläre Sprachen	$\rightsquigarrow$	**Reguläre Sprachen**

Informatik im Jahr 2100: Eine Vorhersage

Java 7	$\rightsquigarrow$	**Taiwan (Red Dragon Edition)**
Python	$\rightsquigarrow$	**Gnu-Boa++# 18.3**
XML	$\rightsquigarrow$	**Medea**
Reguläre Sprachen	$\rightsquigarrow$	**Reguläre Sprachen**
Berechenbare Funktionen	$\rightsquigarrow$	

Informatik im Jahr 2100: Eine Vorhersage

Java 7	$\rightsquigarrow$	**Taiwan (Red Dragon Edition)**
Python	$\rightsquigarrow$	**Gnu-Boa++# 18.3**
XML	$\rightsquigarrow$	**Medea**
Reguläre Sprachen	$\rightsquigarrow$	**Reguläre Sprachen**
Berechenbare Funktionen	$\rightsquigarrow$	**Berechenbare Funktionen**

Informatik im Jahr 2100: Eine Vorhersage

Java 7	$\rightsquigarrow$	**Taiwan (Red Dragon Edition)**
Python	$\rightsquigarrow$	**Gnu-Boa++# 18.3**
XML	$\rightsquigarrow$	**Medea**
Reguläre Sprachen	$\rightsquigarrow$	**Reguläre Sprachen**
Berechenbare Funktionen	$\rightsquigarrow$	**Berechenbare Funktionen**
P	$\rightsquigarrow$	

Informatik im Jahr 2100: Eine Vorhersage

Java 7	$\rightsquigarrow$	**Taiwan (Red Dragon Edition)**
Python	$\rightsquigarrow$	**Gnu-Boa++# 18.3**
XML	$\rightsquigarrow$	**Medea**
Reguläre Sprachen	$\rightsquigarrow$	**Reguläre Sprachen**
Berechenbare Funktionen	$\rightsquigarrow$	**Berechenbare Funktionen**
P	$\rightsquigarrow$	**P**