Entscheidbarkeit und Berechenbarkeit

Worum geht's?

- **Entscheidbarkeit**: Welche Art von Problemen können wir prinzipiell mit Computern lösen?

Worum geht's?

- **Entscheidbarkeit**: Welche Art von Problemen können wir prinzipiell mit Computern lösen?
  - Was heißt prinzipiell?
  - Wir erlauben unbeschränkte Rechenzeit!

Worum geht's?

- **Entscheidbarkeit**: Welche Art von Problemen können wir prinzipiell mit Computern lösen?
  - Was heißt prinzipiell?
  - Wir erlauben unbeschränkte Rechenzeit!
- **Unentscheidbarkeit**: Welche Probleme sind zu schwierig, können also durch Rechner auch in zukünftigen Technologien nicht gelöst werden?

Worum geht's?

- **Entscheidbarkeit**: Welche Art von Problemen können wir prinzipiell mit Computern lösen?
  - Was heißt prinzipiell?
  - Wir erlauben unbeschränkte Rechenzeit!
- **Unentscheidbarkeit**: Welche Probleme sind zu schwierig, können also durch Rechner auch in zukünftigen Technologien nicht gelöst werden?

Wir untersuchen entscheidbare und unentscheidbare Probleme.

Heute, morgen oder immer?

Unsere Aussagen sollten unabhängig von Technologie sein!

- Wir modellieren Rechner wieder durch Turingmaschinen.

Heute, morgen oder immer?

Unsere Aussagen sollten unabhängig von Technologie sein!

- Wir modellieren Rechner wieder durch Turingmaschinen.
- Die Klasse $\mathtt{P}$ und effizient lösbare Probleme:
  - Die Faktorisierung hat wahrscheinlich nur deterministische Berechnungen mit exponentieller Laufzeit, während effiziente Quantenberechnungen möglich sind.
  - Aber: Alles, was effizient mit **deterministischen** Berechnungen in der Zukunft lösbar ist, stimmt wahrscheinlich mit $\mathtt{P}$ überein.

Heute, morgen oder immer?

Unsere Aussagen sollten unabhängig von Technologie sein!

- Wir modellieren Rechner wieder durch Turingmaschinen.
- Die Klasse $\mathtt{P}$ und effizient lösbare Probleme:
  - Die Faktorisierung hat wahrscheinlich nur deterministische Berechnungen mit exponentieller Laufzeit, während effiziente Quantenberechnungen möglich sind.
  - Aber: Alles, was effizient mit **deterministischen** Berechnungen in der Zukunft lösbar ist, stimmt wahrscheinlich mit $\mathtt{P}$ überein.
- Wenn wir die Bedingung der Effizienz fallenlassen, können wir dann

  alles, was auf einem zukünftigen Rechner lösbar ist

  durch

  alles was auf einer deterministischen Turingmaschine lösbar ist

  definieren?

Berechenbare Funktionen

Berechenbare Funktionen

Eine partiell definierte Funktion $f : \Sigma_1^* \to \Sigma_2^*$ heißt genau dann berechenbar, wenn es eine Turingmaschine M gibt, die

- für jede Eingabe $x \in \Sigma_1^*$, auf der $f(x)$ definiert ist, hält und die Ausgabe $f(x)$ produziert.
- Für jede Eingabe $x \in \Sigma_1^*$, für die f nicht definiert ist (also $f(x) = \perp$ gilt), hält M nicht.

Berechenbar oder nicht?

- Die Funktion $f : \{1\}^* \to \{1\}^*$ mit $f(1^n) := 1^{2^n}$ für alle $n \in \mathbb{N}$ ist

Berechenbar oder nicht?

- Die Funktion $f : \{1\}^* \to \{1\}^*$ mit $f(1^n) := 1^{2^n}$ für alle $n \in \mathbb{N}$ ist berechenbar.
- Die überall undefinierte Funktion $f_\emptyset$ ist

Berechenbar oder nicht?

- Die Funktion $f : \{1\}^* \to \{1\}^*$ mit $f(1^n) := 1^{2^n}$ für alle $n \in \mathbb{N}$ ist berechenbar.
- Die überall undefinierte Funktion $f_\emptyset$ ist berechenbar durch einen Algorithmus, der nur aus einer Endlosschleife besteht.

Berechenbar oder nicht?

- Die Funktion $f : \{1\}^* \to \{1\}^*$ mit $f(1^n) := 1^{2^n}$ für alle $n \in \mathbb{N}$ ist berechenbar.
- Die überall undefinierte Funktion $f_\emptyset$ ist berechenbar durch einen Algorithmus, der nur aus einer Endlosschleife besteht.
- Die Funktion g, die überprüft, ob ein ungerichteter Graph zusammenhängend ist, ist

Berechenbar oder nicht?

- Die Funktion $f : \{1\}^* \to \{1\}^*$ mit $f(1^n) := 1^{2^n}$ für alle $n \in \mathbb{N}$ ist berechenbar.
- Die überall undefinierte Funktion $f_\emptyset$ ist berechenbar durch einen Algorithmus, der nur aus einer Endlosschleife besteht.
- Die Funktion g, die überprüft, ob ein ungerichteter Graph zusammenhängend ist, ist berechenbar.
- Die Funktion h, die überprüft, ob ein Programm $\langle M \rangle$ auf Eingabe w hält, ist

Berechenbar oder nicht?

- Die Funktion $f : \{1\}^* \to \{1\}^*$ mit $f(1^n) := 1^{2^n}$ für alle $n \in \mathbb{N}$ ist berechenbar.
- Die überall undefinierte Funktion $f_\emptyset$ ist berechenbar durch einen Algorithmus, der nur aus einer Endlosschleife besteht.
- Die Funktion g, die überprüft, ob ein ungerichteter Graph zusammenhängend ist, ist berechenbar.
- Die Funktion h, die überprüft, ob ein Programm $\langle M \rangle$ auf Eingabe w hält, ist –wie wir später sehen werden– nicht berechenbar.

Entscheidbare Probleme

Entscheidbarkeit

Ein Entscheidungsproblem $L \subseteq \Sigma^*$ heißt genau dann entscheidbar, wenn es eine Turingmaschine M gibt, die für jede Eingabe $x \in \Sigma^*$ hält, so dass

$$x \in L \Leftrightarrow M \text{ akzeptiert } x$$

gilt.

Die Church-Turing These

Die Church-Turing These:

Es gibt genau dann ein stets haltendes „Rechenverfahren“ zur Berechnung einer Funktion oder zur Lösung eines Entscheidungsproblems, wenn es eine Turingmaschine gibt, die die Funktion berechnet, bzw. das Problem löst.

Die Church-Turing These:

Es gibt genau dann ein stets haltendes „Rechenverfahren“ zur Berechnung einer Funktion oder zur Lösung eines Entscheidungsproblems, wenn es eine Turingmaschine gibt, die die Funktion berechnet, bzw. das Problem löst.

- Die Church-Turing These ist nicht beweisbar, sondern nur widerlegbar:

  Der Begriff „Rechenverfahren“ ist nicht definiert.

Die Church-Turing These:

Es gibt genau dann ein stets haltendes „Rechenverfahren“ zur Berechnung einer Funktion oder zur Lösung eines Entscheidungsproblems, wenn es eine Turingmaschine gibt, die die Funktion berechnet, bzw. das Problem löst.

- Die Church-Turing These ist nicht beweisbar, sondern nur widerlegbar:

  Der Begriff „Rechenverfahren“ ist nicht definiert.

Ist das menschliche Genom ein Rechenverfahren, dass die Reaktionen des Körpers auf innere oder äußere Einflüsse, die Eingaben, steuert?

Stimmen unsere Definitionen?

Sind Turingmaschinen mächtig genug?

Stimmen unsere Definitionen?

Sind Turingmaschinen mächtig genug?

Man kann zeigen:

> All das, was in polynomieller Zeit auf einem (deterministischen) parallelen Supercomputer berechnet werden kann, gelingt auch in polynomieller Zeit auf einer Nähmaschine.

Stimmen unsere Definitionen?

Sind Turingmaschinen mächtig genug?

Man kann zeigen:

> All das, was in polynomieller Zeit auf einem (deterministischen) parallelen Supercomputer berechnet werden kann, gelingt auch in polynomieller Zeit auf einer Nähmaschine.

- Deterministische Turingmaschinen überzeugen als Modell eines *deterministischen* Rechners,

Stimmen unsere Definitionen?

Sind Turingmaschinen mächtig genug?

Man kann zeigen:

> All das, was in polynomieller Zeit auf einem (deterministischen) parallelen Supercomputer berechnet werden kann, gelingt auch in polynomieller Zeit auf einer Nähmaschine.

- Deterministische Turingmaschinen überzeugen als Modell eines *deterministischen* Rechners,
- aber können sie auch mit nichtdeterministischen oder probabilistischen Rechnern mithalten?
  Und wie sieht es mit Quantenrechnern aus?

Werden wir jetzt Turingmaschinen programmieren?

- Um Gottes willen, nein!

Werden wir jetzt Turingmaschinen programmieren?

- Um Gottes willen, nein!
- Wir wissen ja, dass Turingmaschinen –bis auf eine polynomielle Verlangsamung– die Kraft moderner Rechner haben und arbeiten

  wie schon in der NP-Vollständigkeit

  nur mit Pseudocode, wenn wir Entscheidbarkeit oder Berechenbarkeit nachweisen wollen.

Welche Probleme sind entscheidbar? (1/2)

Alle Probleme in P und NP sind entscheidbar. Warum?

Welche Probleme sind entscheidbar? (1/2)

Alle Probleme in P und NP sind entscheidbar. Warum?

- Alle bisher behandelten Ordnungs- und Graphprobleme, wie etwa
  - Sortierprobleme,
  - das Zusammenhangsproblem oder
  - das kürzeste-Wege Problem

  gehören zu P und sind deshalb entscheidbar.

Welche Probleme sind entscheidbar? (1/2)

Alle Probleme in P und NP sind entscheidbar. Warum?

- Alle bisher behandelten Ordnungs- und Graphprobleme, wie etwa
  - Sortierprobleme,
  - das Zusammenhangsproblem oder
  - das kürzeste-Wege Problem

  gehören zu P und sind deshalb entscheidbar.
  - Die Entscheidungsversion des längste-Wege Problems ist NP-vollständig, gehört deshalb wahrscheinlich nicht zu P, ist aber entscheidbar.

Welche Probleme sind entscheidbar? (1/2)

Alle Probleme in P und NP sind entscheidbar. Warum?

- Alle bisher behandelten Ordnungs- und Graphprobleme, wie etwa
  - Sortierprobleme,
  - das Zusammenhangsproblem oder
  - das kürzeste-Wege Problem

  gehören zu P und sind deshalb entscheidbar.
  - Die Entscheidungsversion des längste-Wege Problems ist NP-vollständig, gehört deshalb wahrscheinlich nicht zu P, ist aber entscheidbar.
- Die Entscheidungsversion der linearen Programmierung

$$\{(A, b, c, t) \mid \text{Es gibt } x \text{ mit } A \cdot x \leqslant b, x \geqslant \text{ und } x^t \cdot c \geqslant t\}$$

  gehört zu P.

Welche Probleme sind entscheidbar? (1/2)

Alle Probleme in P und NP sind entscheidbar. Warum?

- Alle bisher behandelten Ordnungs- und Graphprobleme, wie etwa
  - Sortierprobleme,
  - das Zusammenhangsproblem oder
  - das kürzeste-Wege Problem

  gehören zu P und sind deshalb entscheidbar.
  - Die Entscheidungsversion des längste-Wege Problems ist NP-vollständig, gehört deshalb wahrscheinlich nicht zu P, ist aber entscheidbar.
- Die Entscheidungsversion der linearen Programmierung

$$\{(A, b, c, t) \mid \text{Es gibt } x \text{ mit } A \cdot x \leqslant b, x \geqslant \text{ und } x^t \cdot c \geqslant t\}$$

gehört zu P. Die 0-1 Programmierung ist entscheidbar, liegt aber wahrscheinlich nicht in P: Es wird $x_1, \ldots, x_n \in \{0, 1\}$ gefordert.

Welche Probleme sind entscheidbar? (2/2)

- Erst vor wenigen Jahren wurde gezeigt, dass das Primzahlproblem

$$\{N \mid N \text{ ist eine Primzahl}\}$$

Welche Probleme sind entscheidbar? (2/2)

- Erst vor wenigen Jahren wurde gezeigt, dass das Primzahlproblem

$$\{N \mid N \text{ ist eine Primzahl } \}$$

zu $\mathtt{P}$ gehört.

N ist eine Binärzahl: die Eingabelänge ist $n = \log_2 N$.

Welche Probleme sind entscheidbar? (2/2)

- Erst vor wenigen Jahren wurde gezeigt, dass das Primzahlproblem

$$\{N \mid \ N \text{ ist eine Primzahl } \}$$

zu $\mathtt{P}$ gehört.

N ist eine Binärzahl: die Eingabelänge ist $n = \log_2 N$.

- Das Faktorisierungsproblem

$$\{(N, a, b) \mid \ \text{es gibt } x \in [a, b] \text{ und } x \text{ teilt } N \}$$

ist entscheidbar, wird aber nicht in $\mathtt{P}$ liegen,

Welche Probleme sind entscheidbar? (2/2)

- Erst vor wenigen Jahren wurde gezeigt, dass das Primzahlproblem

$$\{N \mid \ N \text{ ist eine Primzahl } \}$$

zu $\mathtt{P}$ gehört.

N ist eine Binärzahl: die Eingabelänge ist $n = \log_2 N$.

- Das Faktorisierungsproblem

$$\{(N, a, b) \mid \ \text{es gibt } x \in [a, b] \text{ und } x \text{ teilt } N \}$$

ist entscheidbar, wird aber nicht in $\mathtt{P}$ liegen, zumindest hoffen dies die Kryptographen.

Ist die Erde eine Scheibe?

Stimmt die Klasse NP mit der Klasse entscheidbarer Probleme überein?

Ist die Erde eine Scheibe?

Stimmt die Klasse NP mit der Klasse entscheidbarer Probleme überein?

Das Tautologieproblem

Eingabe: Eine DNF α.

Frage: Sind alle Belegungen von α erfüllend?

Ist die Erde eine Scheibe?

Stimmt die Klasse NP mit der Klasse entscheidbarer Probleme überein?

Das Tautologieproblem

Eingabe: Eine DNF α.

Frage: Sind alle Belegungen von α erfüllend?

Aber natürlich (!?) ist das Tautologieproblem entscheidbar.

PSPACE

PSPACE

Die Klasse PSPACE besteht aus allen Entscheidungsproblemen,

die von deterministischen Turingmaschinem auf polynomiellem Speicherplatz gelöst werden können.

PSPACE

Die Klasse PSPACE besteht aus allen Entscheidungsproblemen, die von deterministischen Turingmaschinem auf polynomiellem Speicherplatz gelöst werden können.

- Jedes Problem $L \in \texttt{PSPACE}$ ist entscheidbar, da es von einer deterministischen Turingmaschine gelöst wird.

PSPACE

Die Klasse PSPACE besteht aus allen Entscheidungsproblemen, die von deterministischen Turingmaschinem auf polynomiellem Speicherplatz gelöst werden können.

- Jedes Problem $L \in \text{PSPACE}$ ist entscheidbar, da es von einer deterministischen Turingmaschine gelöst wird.
- $\text{NP} \subseteq \text{PSPACE}$:

PSPACE

Die Klasse PSPACE besteht aus allen Entscheidungsproblemen, die von deterministischen Turingmaschinem auf polynomiellem Speicherplatz gelöst werden können.

- Jedes Problem $L \in \text{PSPACE}$ ist entscheidbar, da es von einer deterministischen Turingmaschine gelöst wird.
- $\text{NP} \subseteq \text{PSPACE}$:

  Simuliere eine nichtdeterministische Turingmaschine, die in Zeit $t(n)$ rechnet durch eine deterministische Turingmaschine mit Speicherplatz $O(t(n))$.

Wahre quantifizierte boolesche Formeln

QBF

Eingabe: Eine Formel $\alpha \equiv Q_{x_1} \cdots Q_{x_n} \beta(x_1, \ldots x_n)$ mit Existenz- oder Allquantoren $Q_{x_1}, \ldots, Q_{x_n}$ und einer KNF-Formel β.

Frage: Ist α wahr?

Wahre quantifizierte boolesche Formeln

QBF

Eingabe: Eine Formel $\alpha \equiv Q_{x_1} \cdots Q_{x_n} \beta(x_1, \ldots x_n)$ mit Existenz- oder Allquantoren $Q_{x_1}, \ldots, Q_{x_n}$ und einer KNF-Formel β.

Frage: Ist α wahr?

- Ist

$$\alpha_1 \equiv \forall x \exists y \, [\, (\neg x \vee y) \wedge (x \vee \neg y) \,]$$

wahr?

Wahre quantifizierte boolesche Formeln

QBF

Eingabe: Eine Formel $\alpha \equiv Q_{x_1} \cdots Q_{x_n} \beta(x_1, \ldots x_n)$ mit Existenz- oder Allquantoren $Q_{x_1}, \ldots, Q_{x_n}$ und einer KNF-Formel β.

Frage: Ist α wahr?

- Ist

$$\alpha_1 \equiv \forall x \exists y \, [\, (\neg x \vee y) \wedge (x \vee \neg y) \,]$$

wahr? Ja, also gilt $\alpha_1 \in \mathrm{QBF}$.

- Ist

$$\alpha_2 \equiv \exists y \forall x \, [\, (\neg x \vee y) \wedge (x \vee \neg y) \,]$$

wahr?

Wahre quantifizierte boolesche Formeln

QBF

Eingabe: Eine Formel $\alpha \equiv Q_{x_1} \cdots Q_{x_n} \beta(x_1, \ldots x_n)$ mit Existenz- oder Allquantoren $Q_{x_1}, \ldots, Q_{x_n}$ und einer KNF-Formel β.

Frage: Ist α wahr?

- Ist

$$\alpha_1 \equiv \forall x \exists y \, [\, (\neg x \vee y) \wedge (x \vee \neg y) \,]$$

wahr? Ja, also gilt $\alpha_1 \in \mathrm{QBF}$.

- Ist

$$\alpha_2 \equiv \exists y \forall x \, [\, (\neg x \vee y) \wedge (x \vee \neg y) \,]$$

wahr? Nein, also gehört α_2 nicht zu QBF.

Wahre quantifizierte boolesche Formeln

QBF

Eingabe: Eine Formel $\alpha \equiv Q_{x_1} \cdots Q_{x_n} \beta(x_1, \ldots x_n)$ mit Existenz- oder Allquantoren $Q_{x_1}, \ldots, Q_{x_n}$ und einer KNF-Formel β.

Frage: Ist α wahr?

- Ist

$$\alpha_1 \equiv \forall x \exists y \, [\, (\neg x \vee y) \wedge (x \vee \neg y) \,]$$

wahr? Ja, also gilt $\alpha_1 \in \mathrm{QBF}$.

- Ist

$$\alpha_2 \equiv \exists y \forall x \, [\, (\neg x \vee y) \wedge (x \vee \neg y) \,]$$

wahr? Nein, also gehört α_2 nicht zu QBF.

QBF ist auf polynomiellen Platz berechenbar, ist also entscheidbar.

Und mehr als polynomieller Speicherplatz?

- Wenn $S(n)$ stärker wächst als $s(n)$ (wenn also $s(n) = o(S(n))$), dann werden mehr Probleme in Speicherplatz $S(n)$ gelöst als in Speicherplatz $s(n)$.

Und mehr als polynomieller Speicherplatz?

- Wenn $S(n)$ stärker wächst als $s(n)$ (wenn also $s(n) = o(S(n))$), dann werden mehr Probleme in Speicherplatz $S(n)$ gelöst als in Speicherplatz $s(n)$.

  Größerer Speicherplatz liefert größere Berechnungskraft!

Und mehr als polynomieller Speicherplatz?

- Wenn $S(n)$ stärker wächst als $s(n)$ (wenn also $s(n) = o(S(n))$), dann werden mehr Probleme in Speicherplatz $S(n)$ gelöst als in Speicherplatz $s(n)$.

  Größerer Speicherplatz liefert größere Berechnungskraft!
- Auf Speicherplatz 2^n werden Probleme gelöst, die nicht in PSPACE liegen.

  Alle in Speicherplatz 2^n lösbaren Probleme sind entscheidbar. Wir erhalten neue entscheidbare Probleme!

Und mehr als polynomieller Speicherplatz?

- Wenn $S(n)$ stärker wächst als $s(n)$ (wenn also $s(n) = o(S(n))$), dann werden mehr Probleme in Speicherplatz $S(n)$ gelöst als in Speicherplatz $s(n)$.

  Größerer Speicherplatz liefert größere Berechnungskraft!
- Auf Speicherplatz 2^n werden Probleme gelöst, die nicht in PSPACE liegen.

  Alle in Speicherplatz 2^n lösbaren Probleme sind entscheidbar. Wir erhalten neue entscheidbare Probleme!
- Und für Speicherplatz 2^{2^n}?

  Weitere neue entscheidbare Probleme werden gelöst.

Und mehr als polynomieller Speicherplatz?

- Wenn $S(n)$ stärker wächst als $s(n)$ (wenn also $s(n) = o(S(n))$), dann werden mehr Probleme in Speicherplatz $S(n)$ gelöst als in Speicherplatz $s(n)$.

  Größerer Speicherplatz liefert größere Berechnungskraft!
- Auf Speicherplatz 2^n werden Probleme gelöst, die nicht in PSPACE liegen.

  Alle in Speicherplatz 2^n lösbaren Probleme sind entscheidbar. Wir erhalten neue entscheidbare Probleme!
- Und für Speicherplatz 2^{2^n}?

  Weitere neue entscheidbare Probleme werden gelöst.

Die Klasse entscheidbarer Probleme ist riesig!

Church-Turing These: Probabilistische Rechnungen

Eine probabilistische Turingmaschine M hat –wie nichtdeterministische Turingmaschinen– die Wahl unter verschiedenen Befehlen.

Church-Turing These: Probabilistische Rechnungen

Eine probabilistische Turingmaschine M hat –wie nichtdeterministische Turingmaschinen– die Wahl unter verschiedenen Befehlen.

- Jeder anwendbare Befehl b erhält eine Wahrscheinlichkeit p_b gewählt zu werden.

Church-Turing These: Probabilistische Rechnungen

Eine probabilistische Turingmaschine M hat –wie nichtdeterministische Turingmaschinen– die Wahl unter verschiedenen Befehlen.

- Jeder anwendbare Befehl b erhält eine Wahrscheinlichkeit p_b gewählt zu werden.
- Damit erhält jede Berechnung B eine Wahrscheinlichkeit p_B.

Church-Turing These: Probabilistische Rechnungen

Eine probabilistische Turingmaschine M hat –wie nichtdeterministische Turingmaschinen– die Wahl unter verschiedenen Befehlen.

- Jeder anwendbare Befehl b erhält eine Wahrscheinlichkeit p_b gewählt zu werden.
- Damit erhält jede Berechnung B eine Wahrscheinlichkeit p_B.
- M akzeptiert eine Eingabe w genau dann, wenn w mit Wahrscheinlichkeit
  - mindestens $3/4$ (beschränkter Fehler höchstens $1/4$) oder
  - $> 1/2$ (unbeschränkter Fehler $< 1/2$) akzeptiert wird.

Church-Turing These: Probabilistische Rechnungen

Eine probabilistische Turingmaschine M hat –wie nichtdeterministische Turingmaschinen– die Wahl unter verschiedenen Befehlen.

- Jeder anwendbare Befehl b erhält eine Wahrscheinlichkeit p_b gewählt zu werden.
- Damit erhält jede Berechnung B eine Wahrscheinlichkeit p_B.
- M akzeptiert eine Eingabe w genau dann, wenn w mit Wahrscheinlichkeit
  - mindestens $3/4$ (beschränkter Fehler höchstens $1/4$) oder
  - $> 1/2$ (unbeschränkter Fehler $< 1/2$) akzeptiert wird.

Eine probabilistische Turingmaschine (sogar mit unbeschränktem Fehler) und Laufzeit $t(n)$ kann durch eine deterministische Turingmaschine mit Speicherplatz $O(t(n))$ simuliert werden.

Church-Turing These: Und Quanten-Rechnungen?

Eine Quanten-Turingmaschine mit Laufzeit $t(n)$ kann durch eine deterministische Turingmaschine mit Speicherplatz $O(t^2(n))$ simuliert werden.

Church-Turing These: Und Quanten-Rechnungen?

Eine Quanten-Turingmaschine mit Laufzeit $t(n)$ kann durch eine deterministische Turingmaschine mit Speicherplatz $O(t^2(n))$ simuliert werden.

- Starke Indizien für die Church-Turing These!

Church-Turing These: Und Quanten-Rechnungen?

Eine Quanten-Turingmaschine mit Laufzeit $t(n)$ kann durch eine deterministische Turingmaschine mit Speicherplatz $O(t^2(n))$ simuliert werden.

- Starke Indizien für die Church-Turing These!
- Der Begriff der Entscheidbarkeit ändert sich nicht, wenn
  - deterministische,
  - nichtdeterministische,
  - probabilistische oder
  - Quanten-Rechenverfahren

  benutzt werden.

Unentscheidbarkeit

Die meisten Probleme sind unentscheidbar!

Eine Menge X ist (höchstens) abzählbar unendlich, wenn es eine injektive Funktion

$$f : X \to \mathbb{N}$$

gibt. Sonst heißt X überabzählbar.

Die meisten Probleme sind unentscheidbar!

Eine Menge X ist (höchstens) abzählbar unendlich, wenn es eine injektive Funktion

$$f : X \to \mathbb{N}$$

gibt. Sonst heißt X überabzählbar.

- Jede Turingmaschine hat ein endliches Programm und kann eindeutig durch eine natürliche Zahl kodiert werden.

Die meisten Probleme sind unentscheidbar!

Eine Menge X ist (höchstens) abzählbar unendlich, wenn es eine injektive Funktion

$$f : X \to \mathbb{N}$$

gibt. Sonst heißt X überabzählbar.

- Jede Turingmaschine hat ein endliches Programm und kann eindeutig durch eine natürliche Zahl kodiert werden.

  Es gibt nur abzählbar unendlich viele Turingmaschinen.

Die meisten Probleme sind unentscheidbar!

Eine Menge X ist (höchstens) abzählbar unendlich, wenn es eine injektive Funktion

$$f : X \to \mathbb{N}$$

gibt. Sonst heißt X überabzählbar.

- Jede Turingmaschine hat ein endliches Programm und kann eindeutig durch eine natürliche Zahl kodiert werden.

  Es gibt nur abzählbar unendlich viele Turingmaschinen.
- Es gibt überabzählbar viele Entscheidungsprobleme (z.B. über $\Sigma = \{0, 1\}$).

Die meisten Probleme sind unentscheidbar!

Eine Menge X ist (höchstens) abzählbar unendlich, wenn es eine injektive Funktion

$$f : X \to \mathbb{N}$$

gibt. Sonst heißt X überabzählbar.

- Jede Turingmaschine hat ein endliches Programm und kann eindeutig durch eine natürliche Zahl kodiert werden.

  Es gibt nur abzählbar unendlich viele Turingmaschinen.
- Es gibt überabzählbar viele Entscheidungsprobleme (z.B. über $\Sigma = \{0, 1\}$).
  - Für ein Problem L setze $\text{bit}(w) = 1$, falls $w \in L$, und sonst $\text{bit}(w) = 0$.

Die meisten Probleme sind unentscheidbar!

Eine Menge X ist (höchstens) abzählbar unendlich, wenn es eine injektive Funktion

$$f : X \to \mathbb{N}$$

gibt. Sonst heißt X überabzählbar.

- Jede Turingmaschine hat ein endliches Programm und kann eindeutig durch eine natürliche Zahl kodiert werden.

  Es gibt nur abzählbar unendlich viele Turingmaschinen.
- Es gibt überabzählbar viele Entscheidungsprobleme (z.B. über $\Sigma = \{0, 1\}$).
  - Für ein Problem L setze $\text{bit}(w) = 1$, falls $w \in L$, und sonst $\text{bit}(w) = 0$.
  - Fasse den unendlich langen Vektor $(\text{bit}(w) \mid w \in \{0, 1\}^*)$ als die Binärdarstellung einer reellen Zahl im Intervall $[0, 1]$ auf.

Die meisten Probleme sind unentscheidbar!

Eine Menge X ist (höchstens) abzählbar unendlich, wenn es eine injektive Funktion

$$f : X \to \mathbb{N}$$

gibt. Sonst heißt X überabzählbar.

- Jede Turingmaschine hat ein endliches Programm und kann eindeutig durch eine natürliche Zahl kodiert werden.

  Es gibt nur abzählbar unendlich viele Turingmaschinen.
- Es gibt überabzählbar viele Entscheidungsprobleme (z.B. über $\Sigma = \{0, 1\}$).
  - Für ein Problem L setze $\text{bit}(w) = 1$, falls $w \in L$, und sonst $\text{bit}(w) = 0$.
  - Fasse den unendlich langen Vektor $(\text{bit}(w) \mid w \in \{0, 1\}^*)$ als die Binärdarstellung einer reellen Zahl im Intervall $[0, 1]$ auf.
  - Es gibt soviele Entscheidungsprobleme wie reelle Zahlen.

Und nun?

Die meisten Entscheidungsprobleme sind unentscheidbar, aber wahrscheinlich doch auch völlig uninteressant!

Und nun?

Die meisten Entscheidungsprobleme sind unentscheidbar, aber wahrscheinlich doch auch völlig uninteressant!

Wie sehen denn interessante unentscheidbare Probleme aus?

Die Gödelnummer einer Turingmaschine M

O.B.d.A Annahmen über M:

1. M benutzt das Eingabealphabet $\Sigma = \{0, 1\}$ und das Bandalphabet $\Gamma = \{0, 1, \mathrm{B}\}$.

Die Gödelnummer einer Turingmaschine M

O.B.d.A Annahmen über M:

1. M benutzt das Eingabealphabet $\Sigma = \{0, 1\}$ und das Bandalphabet $\Gamma = \{0, 1, \mathrm{B}\}$.
2. Desweiteren hat M die Zustandsmenge $Q = \{0, 1, \ldots, |Q| - 1\}$ und Zustand 1 ist der einzige akzeptierende Zustand.

Die Gödelnummer einer Turingmaschine M

O.B.d.A Annahmen über M:

1. M benutzt das Eingabealphabet $\Sigma = \{0, 1\}$ und das Bandalphabet $\Gamma = \{0, 1, \mathrm{B}\}$.
2. Desweiteren hat M die Zustandsmenge $Q = \{0, 1, \ldots, |Q| - 1\}$ und Zustand 1 ist der einzige akzeptierende Zustand.
3. Anfangszustand ist 0.

Die Gödelnummer einer Turingmaschine M

O.B.d.A Annahmen über M:

1. M benutzt das Eingabealphabet $\Sigma = \{0, 1\}$ und das Bandalphabet $\Gamma = \{0, 1, \mathrm{B}\}$.
2. Desweiteren hat M die Zustandsmenge $Q = \{0, 1, \ldots, |Q| - 1\}$ und Zustand 1 ist der einzige akzeptierende Zustand.
3. Anfangszustand ist 0.

Wie können wir die Maschine M in eine Zahl kodieren, so dass das Dekodieren einfach ist?

Die Gödelnummer einer Turingmaschine M

Für Turingmaschine $M = (Q, \Sigma, \delta, q_0, F)$ definieren wir

$$\langle M \rangle = 1^{|Q|}\ 0\ \mathrm{code}(\delta)\ 00$$

als die Gödelnummer von M.

Die Gödelnummer einer Turingmaschine M

Für Turingmaschine $M = (Q, \Sigma, \delta, q_0, F)$ definieren wir

$$\langle M \rangle = 1^{|Q|}\ 0\ \mathrm{code}(\delta)\ 00$$

als die Gödelnummer von M.

- Da $\Sigma = \{0, 1\}$, $q_0 = 0$ und $F = \{1\}$ kodiert die Gödelnummer alle unbekannten Komponenten von M.

Die Gödelnummer einer Turingmaschine M

Für Turingmaschine $M = (Q, \Sigma, \delta, q_0, F)$ definieren wir

$$\langle M \rangle = 1^{|Q|}\ 0\ \mathrm{code}(\delta)\ 00$$

als die Gödelnummer von M.

- Da $\Sigma = \{0, 1\}$, $q_0 = 0$ und $F = \{1\}$ kodiert die Gödelnummer alle unbekannten Komponenten von M.
- Was ist $\mathrm{code}(\delta)$?

Die Gödelnummer einer Turingmaschine M

Für Turingmaschine $M = (Q, \Sigma, \delta, q_0, F)$ definieren wir

$$\langle M \rangle = 1^{|Q|}\ 0\ \mathrm{code}(\delta)\ 00$$

als die Gödelnummer von M.

- Da $\Sigma = \{0, 1\}$, $q_0 = 0$ und $F = \{1\}$ kodiert die Gödelnummer alle unbekannten Komponenten von M.
- Was ist $\mathrm{code}(\delta)$?
  - $\mathrm{code}(\delta)$ entsteht durch **Konkatenation** der Kodierungen der einzelnen Befehle.

Die Gödelnummer einer Turingmaschine M

Für Turingmaschine $M = (Q, \Sigma, \delta, q_0, F)$ definieren wir

$$\langle M \rangle = 1^{|Q|}\ 0\ \mathrm{code}(\delta)\ 00$$

als die Gödelnummer von M.

- Da $\Sigma = \{0, 1\}$, $q_0 = 0$ und $F = \{1\}$ kodiert die Gödelnummer alle unbekannten Komponenten von M.
- Was ist $\mathrm{code}(\delta)$?
  - $\mathrm{code}(\delta)$ entsteht durch **Konkatenation** der Kodierungen der einzelnen Befehle.
  - Der Befehl $\delta(q, a) = (q', b, \mathrm{wohin})$ wird durch

    $$1^{q+1}\ 0\ 1^{\mathrm{zahl}(a)}\ 0\ 1^{q'+1}\ 0\ 1^{\mathrm{zahl}(b)}\ 0\ 1^{\mathrm{zahl}(\mathtt{wohin})}\ 0$$

    kodiert, wobei

Die Gödelnummer einer Turingmaschine M

Für Turingmaschine $M = (Q, \Sigma, \delta, q_0, F)$ definieren wir

$$\langle M \rangle = 1^{|Q|}\ 0\ \mathrm{code}(\delta)\ 00$$

als die Gödelnummer von M.

- Da $\Sigma = \{0, 1\}$, $q_0 = 0$ und $F = \{1\}$ kodiert die Gödelnummer alle unbekannten Komponenten von M.
- Was ist $\mathrm{code}(\delta)$?
  - $\mathrm{code}(\delta)$ entsteht durch **Konkatenation** der Kodierungen der einzelnen Befehle.
  - Der Befehl $\delta(q, a) = (q', b, \mathrm{wohin})$ wird durch

    $$1^{q+1}\ 0\ 1^{\mathrm{zahl}(a)}\ 0\ 1^{q'+1}\ 0\ 1^{\mathrm{zahl}(b)}\ 0\ 1^{\mathrm{zahl}(\mathtt{wohin})}\ 0$$

    kodiert, wobei $\mathrm{zahl}(0) = 1,\ \mathrm{zahl}(1) = 2,\ \mathrm{zahl}(\mathtt{B}) = 3$, und $\mathrm{zahl}(\mathtt{links}) = 1,\ \mathrm{zahl}(\mathtt{rechts}) = 2,\ \mathrm{zahl}(\mathtt{bleib}) = 3$.

Universelle Turingmaschinen

- In der Gödelnummer $\langle M \rangle$ taucht der String $\texttt{00}$ nur am Ende auf.

Universelle Turingmaschinen

- In der Gödelnummer $\langle M \rangle$ taucht der String $\texttt{00}$ nur am Ende auf.
  - Die Gödelnummer ist „selbstbegrenzend“.

Universelle Turingmaschinen

- In der Gödelnummer $\langle M \rangle$ taucht der String $\texttt{00}$ nur am Ende auf.
  - Die Gödelnummer ist „selbstbegrenzend“.
  - In $\langle M \rangle w$ kann der Suffix w ohne Schwierigkeit rekonstruiert werden: $\texttt{00}$ hat die Funktion eines „Kommas“.

Universelle Turingmaschinen

- In der Gödelnummer $\langle M \rangle$ taucht der String 00 nur am Ende auf.
  - Die Gödelnummer ist „selbstbegrenzend“.
  - In $\langle M \rangle w$ kann der Suffix w ohne Schwierigkeit rekonstruiert werden: 00 hat die Funktion eines „Kommas“.
- Eine **universelle Turingmaschine** U mit den folgenden Eigenschaften kann gebaut werden:
  - M akzeptiert die Eingabe w genau dann, wenn U die Eingabe $\langle M \rangle w$ akzeptiert.
  - M hält genau dann auf Eingabe w, wenn U auf Eingabe $\langle M \rangle w$ hält.

Die Diagonalsprache

Die Diagonalsprache (D)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Akzeptiert M die Eingabe $\langle M \rangle$ nicht?

- D besteht nur aus Gödelnummern von Turingmaschinen M.
  - $\langle M \rangle$ gehört genau dann zu D, wenn die Turingmaschine M die Eingabe $\langle M \rangle$ nicht akzeptiert.

Die Diagonalsprache (D)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Akzeptiert M die Eingabe $\langle M \rangle$ nicht?

- D besteht nur aus Gödelnummern von Turingmaschinen M.
  - $\langle M \rangle$ gehört genau dann zu D, wenn die Turingmaschine M die Eingabe $\langle M \rangle$ nicht akzeptiert.
  - Eine Turingmaschine akzeptiert nicht, wenn verworfen wird

Die Diagonalsprache (D)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Akzeptiert M die Eingabe $\langle M \rangle$ nicht?

- D besteht nur aus Gödelnummern von Turingmaschinen M.
  - $\langle M \rangle$ gehört genau dann zu D, wenn die Turingmaschine M die Eingabe $\langle M \rangle$ nicht akzeptiert.
  - Eine Turingmaschine akzeptiert nicht, wenn verworfen wird oder wenn die Maschine **nicht hält**.

Die Diagonalsprache (D)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Akzeptiert M die Eingabe $\langle M \rangle$ nicht?

- D besteht nur aus Gödelnummern von Turingmaschinen M.
  - $\langle M \rangle$ gehört genau dann zu D, wenn die Turingmaschine M die Eingabe $\langle M \rangle$ nicht akzeptiert.
  - Eine Turingmaschine akzeptiert nicht, wenn verworfen wird oder wenn die Maschine **nicht hält**.

D wird unser erstes unentscheidbares Problem sein.
Die Unentscheidbarkeit weiterer wichtiger Probleme wird folgen.

Die Diagonalsprache D ist nicht entscheidbar

Angenommen, D ist entscheidbar.

Die Diagonalsprache D ist nicht entscheidbar

Angenommen, D ist entscheidbar.

- Dann gibt es eine stets haltende Turingmaschine M^*, die das Entscheidungsproblem D löst, d.h für alle Turingmaschinen M gilt:

Die Diagonalsprache D ist nicht entscheidbar

Angenommen, D ist entscheidbar.

- Dann gibt es eine stets haltende Turingmaschine M^*, die das Entscheidungsproblem D löst, d.h für alle Turingmaschinen M gilt:

$$M^* \text{ akzeptiert } \langle M \rangle \quad \Leftrightarrow \quad \langle M \rangle \in D$$

Die Diagonalsprache D ist nicht entscheidbar

Angenommen, D ist entscheidbar.

- Dann gibt es eine stets haltende Turingmaschine M^*, die das Entscheidungsproblem D löst, d.h für alle Turingmaschinen M gilt:

$$\begin{aligned} M^* \text{ akzeptiert } \langle M \rangle &\Leftrightarrow \langle M \rangle \in D \\ &\Leftrightarrow M \text{ akzeptiert } \langle M \rangle \text{ nicht.} \end{aligned}$$

Die Diagonalsprache D ist nicht entscheidbar

Angenommen, D ist entscheidbar.

- Dann gibt es eine stets haltende Turingmaschine M^*, die das Entscheidungsproblem D löst, d.h für alle Turingmaschinen M gilt:

$$\begin{aligned} M^* \text{ akzeptiert } \langle M \rangle &\Leftrightarrow \langle M \rangle \in D \\ &\Leftrightarrow M \text{ akzeptiert } \langle M \rangle \text{ nicht.} \end{aligned}$$

- Und was macht M^* mit seiner eigenen Gödelnummer?

Die Diagonalsprache D ist nicht entscheidbar

Angenommen, D ist entscheidbar.

- Dann gibt es eine stets haltende Turingmaschine M^*, die das Entscheidungsproblem D löst, d.h für alle Turingmaschinen M gilt:

$$\begin{aligned} M^* \text{ akzeptiert } \langle M \rangle &\Leftrightarrow \langle M \rangle \in D \\ &\Leftrightarrow M \text{ akzeptiert } \langle M \rangle \text{ nicht.} \end{aligned}$$

- Und was macht M^* mit seiner eigenen Gödelnummer?

$$M^* \text{ akzeptiert } \langle M^* \rangle \Leftrightarrow M^* \text{ akzeptiert } \langle M^* \rangle \text{ nicht,}$$

Die Diagonalsprache D ist nicht entscheidbar

Angenommen, D ist entscheidbar.

- Dann gibt es eine stets haltende Turingmaschine M^*, die das Entscheidungsproblem D löst, d.h für alle Turingmaschinen M gilt:

$$M^* \text{ akzeptiert } \langle M \rangle \quad \Leftrightarrow \quad \langle M \rangle \in D$$
$$\Leftrightarrow \quad M \text{ akzeptiert } \langle M \rangle \text{ nicht.}$$

- Und was macht M^* mit seiner eigenen Gödelnummer?

$$M^* \text{ akzeptiert } \langle M^* \rangle \quad \Leftrightarrow \quad M^* \text{ akzeptiert } \langle M^* \rangle \text{ nicht,}$$

und wir haben somit einen Widerspruch zur Entscheidbarkeit erhalten!

Die Diagonalisierungsmethode

- Bilde eine unendliche Matrix T,
  - die für jede Turingmaschine M eine Zeile und
  - für jede Gödelnummer $\langle N \rangle$ eine Spalte besitzt.

Die Diagonalisierungsmethode

- Bilde eine unendliche Matrix T,
  - die für jede Turingmaschine M eine Zeile und
  - für jede Gödelnummer $\langle N \rangle$ eine Spalte besitzt.

$$T(M, \langle N \rangle) = \begin{cases} 1 & M \text{ akzeptiert } \langle N \rangle, \\ 0 & \text{sonst.} \end{cases}$$

Die Diagonalisierungsmethode

- Bilde eine unendliche Matrix T,
  - die für jede Turingmaschine M eine Zeile und
  - für jede Gödelnummer $\langle N \rangle$ eine Spalte besitzt.

$$T(M, \langle N \rangle) = \begin{cases} 1 & M \text{ akzeptiert } \langle N \rangle, \\ 0 & \text{sonst.} \end{cases}$$

- Die Diagonalsprache D „flippt“ die Diagonale von T und erzwingt damit, dass D von keiner Turingmaschine M^* entscheidbar ist:

Die Diagonalisierungsmethode

- Bilde eine unendliche Matrix T,
  - die für jede Turingmaschine M eine Zeile und
  - für jede Gödelnummer $\langle N \rangle$ eine Spalte besitzt.

$$T(M, \langle N \rangle) = \begin{cases} 1 & M \text{ akzeptiert } \langle N \rangle, \\ 0 & \text{sonst.} \end{cases}$$

- Die Diagonalsprache D „flippt“ die Diagonale von T und erzwingt damit, dass D von keiner Turingmaschine M^* entscheidbar ist:
  - M^* wird auf Eingabe $\langle M^* \rangle$ die Anwort $T(M^*, \langle M^* \rangle)$ geben,
  - während D die geflippte Antwort verlangt.

Das Komplement der Diagonalsprache ist unentscheidbar

Wenn L unentscheidbar ist, dann ist auch $\overline{L}$, das Komplement von L, unentscheidbar.

Das Komplement der Diagonalsprache ist unentscheidbar

Wenn L unentscheidbar ist, dann ist auch $\overline{L}$, das Komplement von L, unentscheidbar.

- Angenommen, $\overline{L}$ ist entscheidbar.

Das Komplement der Diagonalsprache ist unentscheidbar

Wenn L unentscheidbar ist, dann ist auch $\overline{L}$, das Komplement von L, unentscheidbar.

- Angenommen, $\overline{L}$ ist entscheidbar.
- Also gibt es eine stets haltende Turingmaschine M, die das Komplement $\overline{L}$ erkennt.

Das Komplement der Diagonalsprache ist unentscheidbar

Wenn L unentscheidbar ist, dann ist auch $\overline{L}$, das Komplement von L, unentscheidbar.

- Angenommen, $\overline{L}$ ist entscheidbar.
- Also gibt es eine stets haltende Turingmaschine M, die das Komplement $\overline{L}$ erkennt.
- Mache verwerfende Zustände akzeptierend und umgekehrt:

  Die neue Turingmaschine M' löst das Problem L.

Das Komplement der Diagonalsprache ist unentscheidbar

Wenn L unentscheidbar ist, dann ist auch $\overline{L}$, das Komplement von L, unentscheidbar.

- Angenommen, $\overline{L}$ ist entscheidbar.
- Also gibt es eine stets haltende Turingmaschine M, die das Komplement $\overline{L}$ erkennt.
- Mache verwerfende Zustände akzeptierend und umgekehrt:

  Die neue Turingmaschine M' löst das Problem L.

Also ist auch das Komplement der Diagonalsprache unentscheidbar.

Reduktionen

Reduktionen

- Die Diagonalsprache ist ein „künstliches“ Problem.
- Wir erhalten weitere Unenstscheidbarkeitsergebnisse mit Hilfe von Reduktionen:

Reduktionen

- Die Diagonalsprache ist ein „künstliches“ Problem.
- Wir erhalten weitere Unenstscheidbarkeitsergebnisse mit Hilfe von Reduktionen:

Wir sagen, dass ein Entscheidungsproblem L_1 auf ein Entscheidungsproblem L_2 reduzierbar ist ($L_1 \leqslant L_2$) genau dann,

Reduktionen

- Die Diagonalsprache ist ein „künstliches“ Problem.
- Wir erhalten weitere Unenstscheidbarkeitsergebnisse mit Hilfe von Reduktionen:

Wir sagen, dass ein Entscheidungsproblem L_1 auf ein Entscheidungsproblem L_2 reduzierbar ist ($L_1 \leqslant L_2$) genau dann, wenn es eine stets haltende Turingmaschine T gibt, so dass für alle $w \in \Sigma_1^*$ gilt:

$$w \in L_1 \Leftrightarrow T(w) \in L_2.$$

Reduktionen

- Die Diagonalsprache ist ein „künstliches“ Problem.
- Wir erhalten weitere Unenstscheidbarkeitsergebnisse mit Hilfe von Reduktionen:

Wir sagen, dass ein Entscheidungsproblem L_1 auf ein Entscheidungsproblem L_2 reduzierbar ist ($L_1 \leqslant L_2$) genau dann, wenn es eine stets haltende Turingmaschine T gibt, so dass für alle $w \in \Sigma_1^*$ gilt:

$$w \in L_1 \Leftrightarrow T(w) \in L_2.$$

$T(w)$ bezeichnet die Ausgabe von T für Eingabe w. T heißt die *transformierende* Turingmaschine.

Unentscheidbarkeit vererbt sich

Das Problem L sei unentscheidbar.
Wenn $L \leqslant K$, dann ist auch K unentscheidbar.

Unentscheidbarkeit vererbt sich

Das Problem L sei unentscheidbar.
Wenn $L \leqslant K$, dann ist auch K unentscheidbar.

- Angenommen, K ist entscheidbar. Dann gibt eine stets haltende Turingmaschine M, die K löst.

Unentscheidbarkeit vererbt sich

Das Problem L sei unentscheidbar.
Wenn $L \leqslant K$, dann ist auch K unentscheidbar.

- Angenommen, K ist entscheidbar. Dann gibt eine stets haltende Turingmaschine M, die K löst.
- Da $L \leqslant K$, gibt es eine stets haltende Turingmaschine T mit

$$w \in L \Leftrightarrow T(w) \in K.$$

Also gilt $w \in L \Leftrightarrow M$ akzeptiert die Eingabe $T(w)$.

Unentscheidbarkeit vererbt sich

Das Problem L sei unentscheidbar.
Wenn $L \leqslant K$, dann ist auch K unentscheidbar.

- Angenommen, K ist entscheidbar. Dann gibt eine stets haltende Turingmaschine M, die K löst.
- Da $L \leqslant K$, gibt es eine stets haltende Turingmaschine T mit

$$w \in L \Leftrightarrow T(w) \in K.$$

  Also gilt $w \in L \Leftrightarrow$ M akzeptiert die Eingabe $T(w)$.
- Damit wird L aber durch eine stets haltende Turingmaschine, nämlich die Maschine, die zuerst T und dann M simuliert, gelöst.

Unentscheidbarkeit vererbt sich

Das Problem L sei unentscheidbar.
Wenn $L \leqslant K$, dann ist auch K unentscheidbar.

- Angenommen, K ist entscheidbar. Dann gibt eine stets haltende Turingmaschine M, die K löst.
- Da $L \leqslant K$, gibt es eine stets haltende Turingmaschine T mit

$$w \in L \Leftrightarrow T(w) \in K.$$

  Also gilt $w \in L \Leftrightarrow$ M akzeptiert die Eingabe $T(w)$.
- Damit wird L aber durch eine stets haltende Turingmaschine, nämlich die Maschine, die zuerst T und dann M simuliert, gelöst.
- Also ist L im Widerspruch zur Annahme doch entscheidbar.

Ist die Unentscheidbarkeit von D interessant?

- Die Diagonalsprache scheint „künstlich“ zu sein: Sie „redet“ über Turingmaschinen, die ihre eigene Gödelnummer (als Eingabe) nicht akzeptieren.

Ist die Unentscheidbarkeit von D interessant?

- Die Diagonalsprache scheint „künstlich“ zu sein: Sie „redet“ über Turingmaschinen, die ihre eigene Gödelnummer (als Eingabe) nicht akzeptieren.
- Das Komplement $\overline{D}$ redet dann natürlich über Turingmaschinen, die ihre Gödelnummer als Eingabe akzeptieren.

Ist die Unentscheidbarkeit von D interessant?

- Die Diagonalsprache scheint „künstlich“ zu sein: Sie „redet“ über Turingmaschinen, die ihre eigene Gödelnummer (als Eingabe) nicht akzeptieren.
- Das Komplement $\overline{D}$ redet dann natürlich über Turingmaschinen, die ihre Gödelnummer als Eingabe akzeptieren.
  - Na, und? Immer noch nicht interessant!

Ist die Unentscheidbarkeit von D interessant?

- Die Diagonalsprache scheint „künstlich“ zu sein: Sie „redet“ über Turingmaschinen, die ihre eigene Gödelnummer (als Eingabe) nicht akzeptieren.
- Das Komplement $\overline{D}$ redet dann natürlich über Turingmaschinen, die ihre Gödelnummer als Eingabe akzeptieren.
  - Na, und? Immer noch nicht interessant!
  - Aber die Frage,

    ob eine Turingmaschine ihre Gödelnummer als Eingabe akzeptiert,

    ist schon unentscheidbar und leichter als die Frage,

    ob eine Turingmaschine eine beliebige Eingabe akzeptiert.

Ist die Unentscheidbarkeit von D interessant?

- Die Diagonalsprache scheint „künstlich“ zu sein: Sie „redet“ über Turingmaschinen, die ihre eigene Gödelnummer (als Eingabe) nicht akzeptieren.
- Das Komplement $\overline{D}$ redet dann natürlich über Turingmaschinen, die ihre Gödelnummer als Eingabe akzeptieren.
  - Na, und? Immer noch nicht interessant!
  - Aber die Frage,

    ob eine Turingmaschine ihre Gödelnummer als Eingabe akzeptiert,

    ist schon unentscheidbar und leichter als die Frage,

    ob eine Turingmaschine eine beliebige Eingabe akzeptiert.
- Die Frage nach der Akzeptanz einer beliebigen Eingabe ist also unentscheidbar?

Die universelle Sprache

Die universelle Sprache (U)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Akzeptiert M die Eingabe x?

Die universelle Sprache (U)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Akzeptiert M die Eingabe x?

- Wir zeigen die Reduktion $\overline{D} \leqslant U$.
  - Da $\overline{D}$ unentscheidbar ist, ist damit auch U unentscheidbar.

Die universelle Sprache (U)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Akzeptiert M die Eingabe x?

- Wir zeigen die Reduktion $\overline{D} \leqslant U$.
  - Da $\overline{D}$ unentscheidbar ist, ist damit auch U unentscheidbar.
- Die transformierende Turingmaschine T
  - prüft zuerst, ob die Eingabe w für $\overline{D}$ die Gödelnummer einer Turingmaschine M ist, also ob $w = \langle M \rangle$ gilt.

Die universelle Sprache (U)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Akzeptiert M die Eingabe x?

- Wir zeigen die Reduktion $\overline{D} \leqslant U$.
  - Da $\overline{D}$ unentscheidbar ist, ist damit auch U unentscheidbar.
- Die transformierende Turingmaschine T
  - prüft zuerst, ob die Eingabe w für $\overline{D}$ die Gödelnummer einer Turingmaschine M ist, also ob $w = \langle M \rangle$ gilt.
  - Wenn ja, gibt T die Ausgabe $T(w) = \langle M \rangle x$

Die universelle Sprache (U)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Akzeptiert M die Eingabe x?

- Wir zeigen die Reduktion $\overline{D} \leqslant U$.
  - Da $\overline{D}$ unentscheidbar ist, ist damit auch U unentscheidbar.
- Die transformierende Turingmaschine T
  - prüft zuerst, ob die Eingabe w für $\overline{D}$ die Gödelnummer einer Turingmaschine M ist, also ob $w = \langle M \rangle$ gilt.
  - Wenn ja, gibt T die Ausgabe $T(w) = \langle M \rangle x = \langle M \rangle \langle M \rangle$.

Die universelle Sprache (U)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Akzeptiert M die Eingabe x?

- Wir zeigen die Reduktion $\overline{D} \leqslant U$.
  - Da $\overline{D}$ unentscheidbar ist, ist damit auch U unentscheidbar.
- Die transformierende Turingmaschine T
  - prüft zuerst, ob die Eingabe w für $\overline{D}$ die Gödelnummer einer Turingmaschine M ist, also ob $w = \langle M \rangle$ gilt.
  - Wenn ja, gibt T die Ausgabe $T(w) = \langle M \rangle x = \langle M \rangle \langle M \rangle$. Also ist

$$w \in \overline{D} \quad \Leftrightarrow \quad \langle M \rangle \in \overline{D}$$

Die universelle Sprache (U)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Akzeptiert M die Eingabe x?

- Wir zeigen die Reduktion $\overline{D} \leqslant U$.
  - Da $\overline{D}$ unentscheidbar ist, ist damit auch U unentscheidbar.
- Die transformierende Turingmaschine T
  - prüft zuerst, ob die Eingabe w für $\overline{D}$ die Gödelnummer einer Turingmaschine M ist, also ob $w = \langle M \rangle$ gilt.
  - Wenn ja, gibt T die Ausgabe $T(w) = \langle M \rangle x = \langle M \rangle \langle M \rangle$. Also ist

$$\begin{aligned} w \in \overline{D} &\Leftrightarrow \langle M \rangle \in \overline{D} \\ &\Leftrightarrow M \text{ akzeptiert } \langle M \rangle \end{aligned}$$

Die universelle Sprache (U)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Akzeptiert M die Eingabe x?

- Wir zeigen die Reduktion $\overline{D} \leqslant U$.
  - Da $\overline{D}$ unentscheidbar ist, ist damit auch U unentscheidbar.
- Die transformierende Turingmaschine T
  - prüft zuerst, ob die Eingabe w für $\overline{D}$ die Gödelnummer einer Turingmaschine M ist, also ob $w = \langle M \rangle$ gilt.
  - Wenn ja, gibt T die Ausgabe $T(w) = \langle M \rangle x = \langle M \rangle \langle M \rangle$. Also ist

$$\begin{aligned} w \in \overline{D} &\Leftrightarrow \langle M \rangle \in \overline{D} \\ &\Leftrightarrow M \text{ akzeptiert } \langle M \rangle \\ &\Leftrightarrow \langle M \rangle x \in U \end{aligned}$$

Die universelle Sprache (U)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Akzeptiert M die Eingabe x?

- Wir zeigen die Reduktion $\overline{D} \leqslant U$.
  - Da $\overline{D}$ unentscheidbar ist, ist damit auch U unentscheidbar.
- Die transformierende Turingmaschine T
  - prüft zuerst, ob die Eingabe w für $\overline{D}$ die Gödelnummer einer Turingmaschine M ist, also ob $w = \langle M \rangle$ gilt.
  - Wenn ja, gibt T die Ausgabe $T(w) = \langle M \rangle x = \langle M \rangle \langle M \rangle$. Also ist

$$\begin{aligned} w \in \overline{D} &\Leftrightarrow \langle M \rangle \in \overline{D} \\ &\Leftrightarrow M \text{ akzeptiert } \langle M \rangle \\ &\Leftrightarrow \langle M \rangle x \in U \\ &\Leftrightarrow T(w) \in U. \end{aligned}$$

Die universelle Sprache ist unentscheidbar

Und was machen wir, wenn w keine Gödelnummer ist?

- Wenn w keine Gödelnummer ist, dann ist $w \in \overline{D}$.

Die universelle Sprache ist unentscheidbar

Und was machen wir, wenn w keine Gödelnummer ist?

- Wenn w keine Gödelnummer ist, dann ist $w \in \overline{D}$.
- Wir müssen also Sorge tragen, dass $T(w) \in U$ gilt.

Die universelle Sprache ist unentscheidbar

Und was machen wir, wenn w keine Gödelnummer ist?

- Wenn w keine Gödelnummer ist, dann ist $w \in \overline{D}$.
- Wir müssen also Sorge tragen, dass $T(w) \in U$ gilt.
  - T gibt die Ausgabe $T(w) = \langle M_0 \rangle \epsilon$, wobei die Turingmaschine M_0 auf Eingabe ϵ sofort hält und akzeptiert.

Die universelle Sprache ist unentscheidbar

Und was machen wir, wenn w keine Gödelnummer ist?

- Wenn w keine Gödelnummer ist, dann ist $w \in \overline{D}$.
- Wir müssen also Sorge tragen, dass $T(w) \in U$ gilt.
  - T gibt die Ausgabe $T(w) = \langle M_0 \rangle \epsilon$, wobei die Turingmaschine M_0 auf Eingabe ϵ sofort hält und akzeptiert.
  - Aber dann ist $T(w) \in U$ wie gefordert.

Die universelle Sprache ist unentscheidbar

Und was machen wir, wenn w keine Gödelnummer ist?

- Wenn w keine Gödelnummer ist, dann ist $w \in \overline{D}$.
- Wir müssen also Sorge tragen, dass $T(w) \in U$ gilt.
  - T gibt die Ausgabe $T(w) = \langle M_0 \rangle \epsilon$, wobei die Turingmaschine M_0 auf Eingabe ϵ sofort hält und akzeptiert.
  - Aber dann ist $T(w) \in U$ wie gefordert.

Ab jetzt werden wir den Fall unterschlagen, dass die Eingabe ein gefordertes Format nicht besitzt.

Das Halteproblem

Was wäre denn wirklich höchst interessant?

- Wenn wir einen Supercompiler bauen könnten, der
  - brav ein Quellprogramm P compiliert und
  - dann überprüft, ob P für alle Eingaben hält!

Was wäre denn wirklich höchst interessant?

- Wenn wir einen Supercompiler bauen könnten, der
  - brav ein Quellprogramm P compiliert und
  - dann überprüft, ob P für alle Eingaben hält!
- Wir sind weniger ambitioniert und fragen nur, ob P für eine bestimmte Eingabe w hält.

Was wäre denn wirklich höchst interessant?

- Wenn wir einen Supercompiler bauen könnten, der
  - brav ein Quellprogramm P compiliert und
  - dann überprüft, ob P für alle Eingaben hält!
- Wir sind weniger ambitioniert und fragen nur, ob P für eine bestimmte Eingabe w hält.
- Wir wissen, dass die Frage nach der *Akzeptanz* von w unentscheidbar ist.

Was wäre denn wirklich höchst interessant?

- Wenn wir einen Supercompiler bauen könnten, der
  - brav ein Quellprogramm P compiliert und
  - dann überprüft, ob P für alle Eingaben hält!
- Wir sind weniger ambitioniert und fragen nur, ob P für eine bestimmte Eingabe w hält.
- Wir wissen, dass die Frage nach der *Akzeptanz* von w unentscheidbar ist.

Aber dann sollte doch auch die Frage nach dem *Halten* für w unentscheidbar sein und es gibt keinen Supercompiler?

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.
- U redet über Akzeptanz, H nur über die Eigenschaft zu halten.

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.
- U redet über Akzeptanz, H nur über die Eigenschaft zu halten.
  - Die Transformation T muss $w \in U \Leftrightarrow T(w) \in H$ erfüllen.

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.
- U redet über Akzeptanz, H nur über die Eigenschaft zu halten.
  - Die Transformation T muss $w \in U \Leftrightarrow T(w) \in H$ erfüllen.
  - Wenn $w = \langle M \rangle x$, dann

    $w \in U$ genau dann, wenn M Eingabe x akzeptiert.

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.
- U redet über Akzeptanz, H nur über die Eigenschaft zu halten.
  - Die Transformation T muss $w \in U \Leftrightarrow T(w) \in H$ erfüllen.
  - Wenn $w = \langle M \rangle x$, dann

    $w \in U$ genau dann, wenn M Eingabe x akzeptiert.
  - Wenn $T(w) = \langle M^* \rangle x^*$, dann

    $T(w) \in H$ genau dann, wenn M^* auf Eingabe x^* hält.

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.
- U redet über Akzeptanz, H nur über die Eigenschaft zu halten.
  - Die Transformation T muss $w \in U \Leftrightarrow T(w) \in H$ erfüllen.
  - Wenn $w = \langle M \rangle x$, dann

    $w \in U$ genau dann, wenn M Eingabe x akzeptiert.
  - Wenn $T(w) = \langle M^* \rangle x^*$, dann

    $T(w) \in H$ genau dann, wenn M^* auf Eingabe x^* hält.
- Wir lassen x unverändert (also $x^* = x$), modifizieren aber M:

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.
- U redet über Akzeptanz, H nur über die Eigenschaft zu halten.
  - Die Transformation T muss $w \in U \Leftrightarrow T(w) \in H$ erfüllen.
  - Wenn $w = \langle M \rangle x$, dann

    $w \in U$ genau dann, wenn M Eingabe x akzeptiert.
  - Wenn $T(w) = \langle M^* \rangle x^*$, dann

    $T(w) \in H$ genau dann, wenn M^* auf Eingabe x^* hält.
- Wir lassen x unverändert (also $x^* = x$), modifizieren aber M:
  - M^* verhält sich so wie M. Wenn aber M in einem verwerfenden Zustand hält, dann

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.
- U redet über Akzeptanz, H nur über die Eigenschaft zu halten.
  - Die Transformation T muss $w \in U \Leftrightarrow T(w) \in H$ erfüllen.
  - Wenn $w = \langle M \rangle x$, dann

    $w \in U$ genau dann, wenn M Eingabe x akzeptiert.
  - Wenn $T(w) = \langle M^* \rangle x^*$, dann

    $T(w) \in H$ genau dann, wenn M^* auf Eingabe x^* hält.
- Wir lassen x unverändert (also $x^* = x$), modifizieren aber M:
  - M^* verhält sich so wie M. Wenn aber M in einem verwerfenden Zustand hält, dann **zwingen wir M^* in eine Endlosschleife**!

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.
- U redet über Akzeptanz, H nur über die Eigenschaft zu halten.
  - Die Transformation T muss $w \in U \Leftrightarrow T(w) \in H$ erfüllen.
  - Wenn $w = \langle M \rangle x$, dann

    $w \in U$ genau dann, wenn M Eingabe x akzeptiert.
  - Wenn $T(w) = \langle M^* \rangle x^*$, dann

    $T(w) \in H$ genau dann, wenn M^* auf Eingabe x^* hält.
- Wir lassen x unverändert (also $x^* = x$), modifizieren aber M:
  - M^* verhält sich so wie M. Wenn aber M in einem verwerfenden Zustand hält, dann **zwingen wir M^* in eine Endlosschleife**!
  - $\langle M \rangle x \in U \Leftrightarrow M$ akzeptiert x

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.
- U redet über Akzeptanz, H nur über die Eigenschaft zu halten.
  - Die Transformation T muss $w \in U \Leftrightarrow T(w) \in H$ erfüllen.
  - Wenn $w = \langle M \rangle x$, dann

    $w \in U$ genau dann, wenn M Eingabe x akzeptiert.
  - Wenn $T(w) = \langle M^* \rangle x^*$, dann

    $T(w) \in H$ genau dann, wenn M^* auf Eingabe x^* hält.
- Wir lassen x unverändert (also $x^* = x$), modifizieren aber M:
  - M^* verhält sich so wie M. Wenn aber M in einem verwerfenden Zustand hält, dann **zwingen wir M^* in eine Endlosschleife**!
  - $\langle M \rangle x \in U \Leftrightarrow M$ akzeptiert $x \Leftrightarrow M^*$ hält auf x

Das Halteproblem (H)

Eingabe: Eine Gödelnummer $\langle M \rangle$ und eine Eingabe x.

Frage: Hält M auf Eingabe x?

- Wir weisen die Reduktion $U \leqslant H$ nach und das Halteproblem ist deshalb nicht entscheidbar.
- U redet über Akzeptanz, H nur über die Eigenschaft zu halten.
  - Die Transformation T muss $w \in U \Leftrightarrow T(w) \in H$ erfüllen.
  - Wenn $w = \langle M \rangle x$, dann

    $w \in U$ genau dann, wenn M Eingabe x akzeptiert.
  - Wenn $T(w) = \langle M^* \rangle x^*$, dann

    $T(w) \in H$ genau dann, wenn M^* auf Eingabe x^* hält.
- Wir lassen x unverändert (also $x^* = x$), modifizieren aber M:
  - M^* verhält sich so wie M. Wenn aber M in einem verwerfenden Zustand hält, dann **zwingen wir M^* in eine Endlosschleife**!
  - $\langle M \rangle x \in U \Leftrightarrow M$ akzeptiert $x \Leftrightarrow M^*$ hält auf $x \Leftrightarrow \langle M^* \rangle x \in H$.

Das spezielle Halteproblem

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

- Wir weisen die Reduktion $H \leqslant H_\epsilon$ nach und das spezielle Halteproblem ist deshalb nicht entscheidbar.

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

- Wir weisen die Reduktion $H \leqslant H_\epsilon$ nach und das spezielle Halteproblem ist deshalb nicht entscheidbar.
- $w = \langle M \rangle u$ sei eine Eingabe für das Halteproblem. Wenn $T(w) = \langle M^* \rangle$, dann müssen wir zeigen

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

- Wir weisen die Reduktion $H \leqslant H_\epsilon$ nach und das spezielle Halteproblem ist deshalb nicht entscheidbar.
- $w = \langle M \rangle u$ sei eine Eingabe für das Halteproblem. Wenn $T(w) = \langle M^* \rangle$, dann müssen wir zeigen

  $w \in H \Leftrightarrow M$ hält auf x

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

- Wir weisen die Reduktion $H \leqslant H_\epsilon$ nach und das spezielle Halteproblem ist deshalb nicht entscheidbar.
- $w = \langle M \rangle u$ sei eine Eingabe für das Halteproblem. Wenn $T(w) = \langle M^* \rangle$, dann müssen wir zeigen

  $w \in H \;\Leftrightarrow\; M$ hält auf $x \overset{!}{\Leftrightarrow} M^*$ hält auf ϵ

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

- Wir weisen die Reduktion $H \leqslant H_\epsilon$ nach und das spezielle Halteproblem ist deshalb nicht entscheidbar.
- $w = \langle M \rangle u$ sei eine Eingabe für das Halteproblem. Wenn $T(w) = \langle M^* \rangle$, dann müssen wir zeigen

  $w \in H \;\Leftrightarrow\; M$ hält auf $x \overset{!}{\Leftrightarrow} M^*$ hält auf $\epsilon \;\Leftrightarrow\; T(w) \in H_\epsilon$.

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

- Wir weisen die Reduktion $H \leqslant H_\epsilon$ nach und das spezielle Halteproblem ist deshalb nicht entscheidbar.
- $w = \langle M \rangle u$ sei eine Eingabe für das Halteproblem. Wenn $T(w) = \langle M^* \rangle$, dann müssen wir zeigen

  $w \in H \;\Leftrightarrow\; M$ hält auf $x \overset{!}{\Leftrightarrow} M^*$ hält auf $\epsilon \;\Leftrightarrow\; T(w) \in H_\epsilon$.
- Was sollte M^* machen?

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

- Wir weisen die Reduktion $H \leqslant H_\epsilon$ nach und das spezielle Halteproblem ist deshalb nicht entscheidbar.
- $w = \langle M \rangle u$ sei eine Eingabe für das Halteproblem. Wenn $T(w) = \langle M^* \rangle$, dann müssen wir zeigen

  $w \in H \;\Leftrightarrow\; M$ hält auf $x \overset{!}{\Leftrightarrow} M^*$ hält auf $\epsilon \;\Leftrightarrow\; T(w) \in H_\epsilon$.
- Was sollte M^* machen?
  - Wir „speichern“ x in den Befehlen von M^* ab.

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

- Wir weisen die Reduktion $H \leqslant H_\epsilon$ nach und das spezielle Halteproblem ist deshalb nicht entscheidbar.
- $w = \langle M \rangle u$ sei eine Eingabe für das Halteproblem. Wenn $T(w) = \langle M^* \rangle$, dann müssen wir zeigen

  $w \in H \;\Leftrightarrow\; M$ hält auf $x \overset{!}{\Leftrightarrow} M^*$ hält auf $\epsilon \;\Leftrightarrow\; T(w) \in H_\epsilon$.
- Was sollte M^* machen?
  - Wir „speichern“ x in den Befehlen von M^* ab.
  - M^* schreibt als erste Aktion x auf das leere Band und

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

- Wir weisen die Reduktion $H \leqslant H_\epsilon$ nach und das spezielle Halteproblem ist deshalb nicht entscheidbar.
- $w = \langle M \rangle u$ sei eine Eingabe für das Halteproblem. Wenn $T(w) = \langle M^* \rangle$, dann müssen wir zeigen

  $w \in H \;\Leftrightarrow\; M$ hält auf $x \overset{!}{\Leftrightarrow} M^*$ hält auf $\epsilon \;\Leftrightarrow\; T(w) \in H_\epsilon$.
- Was sollte M^* machen?
  - Wir „speichern“ x in den Befehlen von M^* ab.
  - M^* schreibt als erste Aktion x auf das leere Band und
  - simuliert dann M (auf der jetzigen Eingabe x).

Das spezielle Halteproblem (H_ϵ)

Eingabe: Eine Gödelnummer $\langle M \rangle$.

Frage: Hält M auf dem leeren Wort?

- Wir weisen die Reduktion $H \leqslant H_\epsilon$ nach und das spezielle Halteproblem ist deshalb nicht entscheidbar.
- $w = \langle M \rangle u$ sei eine Eingabe für das Halteproblem. Wenn $T(w) = \langle M^* \rangle$, dann müssen wir zeigen

  $w \in H \;\Leftrightarrow\; M$ hält auf $x \overset{!}{\Leftrightarrow} M^*$ hält auf $\epsilon \;\Leftrightarrow\; T(w) \in H_\epsilon$.
- Was sollte M^* machen?
  - Wir „speichern“ x in den Befehlen von M^* ab.
  - M^* schreibt als erste Aktion x auf das leere Band und
  - simuliert dann M (auf der jetzigen Eingabe x).
- Wie verlangt: M hält auf $x \Leftrightarrow M^*$ hält auf ϵ.

Der Satz von Rice

Die Aufgabe, nicht-triviale Eigenschaften von Programmen festzustellen, ist unentscheidbar.

Der Satz von Rice (1/3)

Wie schwierig sind die folgenden Fragen für ein vorgegebenes Programm P?

- Das Verifikationsproblem: Berechnet P die Funktion f?

Der Satz von Rice (1/3)

Wie schwierig sind die folgenden Fragen für ein vorgegebenes Programm P?

- Das Verifikationsproblem: Berechnet P die Funktion f?
- Hält P für alle Eingaben?

Der Satz von Rice (1/3)

Wie schwierig sind die folgenden Fragen für ein vorgegebenes Programm P?

- Das Verifikationsproblem: Berechnet P die Funktion f?
- Hält P für alle Eingaben?
- Verhält sich P wie ein endlicher Automat?

Der Satz von Rice (1/3)

Wie schwierig sind die folgenden Fragen für ein vorgegebenes Programm P?

- Das Verifikationsproblem: Berechnet P die Funktion f?
- Hält P für alle Eingaben?
- Verhält sich P wie ein endlicher Automat?

- Es stellt sich heraus, dass alle Fragen unentscheidbar sind.

Der Satz von Rice (1/3)

Wie schwierig sind die folgenden Fragen für ein vorgegebenes Programm P?

- Das Verifikationsproblem: Berechnet P die Funktion f?
- Hält P für alle Eingaben?
- Verhält sich P wie ein endlicher Automat?

- Es stellt sich heraus, dass alle Fragen unentscheidbar sind.
- Schlimmer noch, jede Frage nach einer nicht-trivialen Eigenschaft eines Programms ist unentscheidbar:

Der Satz von Rice (1/3)

Wie schwierig sind die folgenden Fragen für ein vorgegebenes Programm P?

- Das Verifikationsproblem: Berechnet P die Funktion f?
- Hält P für alle Eingaben?
- Verhält sich P wie ein endlicher Automat?

- Es stellt sich heraus, dass alle Fragen unentscheidbar sind.
- Schlimmer noch, jede Frage nach einer nicht-trivialen Eigenschaft eines Programms ist unentscheidbar:
  - Sei B_{Σ_1,Σ_2} die Menge aller berechenbaren Funktionen $f : \Sigma_1^* \to \Sigma_2^*$.

Der Satz von Rice (1/3)

Wie schwierig sind die folgenden Fragen für ein vorgegebenes Programm P?

- Das Verifikationsproblem: Berechnet P die Funktion f?
- Hält P für alle Eingaben?
- Verhält sich P wie ein endlicher Automat?

- Es stellt sich heraus, dass alle Fragen unentscheidbar sind.
- Schlimmer noch, jede Frage nach einer nicht-trivialen Eigenschaft eines Programms ist unentscheidbar:
  - Sei B_{Σ_1,Σ_2} die Menge aller berechenbaren Funktionen $f : \Sigma_1^* \to \Sigma_2^*$.
  - Eine Eigenschaft S ist eine Teilmenge $S \subseteq B_{\Sigma_1,\Sigma_2}$.

Der Satz von Rice (1/3)

Wie schwierig sind die folgenden Fragen für ein vorgegebenes Programm P?

- Das Verifikationsproblem: Berechnet P die Funktion f?
- Hält P für alle Eingaben?
- Verhält sich P wie ein endlicher Automat?

- Es stellt sich heraus, dass alle Fragen unentscheidbar sind.
- Schlimmer noch, jede Frage nach einer nicht-trivialen Eigenschaft eines Programms ist unentscheidbar:
  - Sei B_{Σ_1,Σ_2} die Menge aller berechenbaren Funktionen $f : \Sigma_1^* \to \Sigma_2^*$.
  - Eine Eigenschaft S ist eine Teilmenge $S \subseteq B_{\Sigma_1,\Sigma_2}$.
  - Die Eigenschaft S ist nicht-trivial, wenn

Der Satz von Rice (1/3)

Wie schwierig sind die folgenden Fragen für ein vorgegebenes Programm P?

- Das Verifikationsproblem: Berechnet P die Funktion f?
- Hält P für alle Eingaben?
- Verhält sich P wie ein endlicher Automat?

- Es stellt sich heraus, dass alle Fragen unentscheidbar sind.
- Schlimmer noch, jede Frage nach einer nicht-trivialen Eigenschaft eines Programms ist unentscheidbar:
  - Sei B_{Σ_1,Σ_2} die Menge aller berechenbaren Funktionen $f : \Sigma_1^* \to \Sigma_2^*$.
  - Eine Eigenschaft S ist eine Teilmenge $S \subseteq B_{\Sigma_1,\Sigma_2}$.
  - Die Eigenschaft S ist nicht-trivial, wenn $S \neq \emptyset$ und $S \neq B_{\Sigma_1,\Sigma_2}$.

Der Satz von Rice (2/3)

Wenn S nicht-trivial ist, dann ist

$$\mathrm{B}(S) = \{\langle M \rangle \mid \text{die von } M \text{ berechnete Funktion gehört zu } S\}$$

unentscheidbar.

Der Satz von Rice (2/3)

Wenn S nicht-trivial ist, dann ist

$$\mathrm{B}(S) = \{\langle M \rangle \mid \text{die von } M \text{ berechnete Funktion gehört zu } S\}$$

unentscheidbar.

- Wir weisen die Reduktion $\overline{H_\epsilon} \leqslant \mathrm{B}(S)$ nach und $B(S)$ ist deshalb nicht entscheidbar.

Der Satz von Rice (2/3)

Wenn S nicht-trivial ist, dann ist

$$\mathrm{B}(S) = \{\langle M \rangle \mid \text{die von } M \text{ berechnete Funktion gehört zu } S\}$$

unentscheidbar.

- Wir weisen die Reduktion $\overline{H_\epsilon} \leqslant \mathrm{B}(S)$ nach und $B(S)$ ist deshalb nicht entscheidbar.
- `nie` sei die überall auf `nil` gesetzte Funktion. O.B.d.A `nie` $\in S$.

Der Satz von Rice (2/3)

Wenn S nicht-trivial ist, dann ist

$$\mathrm{B}(S) = \{\langle M \rangle \mid \text{die von } M \text{ berechnete Funktion gehört zu } S\}$$

unentscheidbar.

- Wir weisen die Reduktion $\overline{H_\epsilon} \leqslant \mathrm{B}(S)$ nach und $B(S)$ ist deshalb nicht entscheidbar.
- `nie` sei die überall auf `nil` gesetzte Funktion. O.B.d.A `nie` $\in S$.
- Die Transformation $T(\langle M \rangle) = \langle M^* \rangle$ soll erfüllen

$$\langle M \rangle \in \overline{H_\epsilon} \quad \Leftrightarrow \quad M \text{ hält nicht auf } \epsilon$$

Der Satz von Rice (2/3)

Wenn S nicht-trivial ist, dann ist

$$\mathrm{B}(S) = \{\langle M \rangle \mid \text{die von } M \text{ berechnete Funktion gehört zu } S\}$$

unentscheidbar.

- Wir weisen die Reduktion $\overline{H_\epsilon} \leqslant \mathrm{B}(S)$ nach und $B(S)$ ist deshalb nicht entscheidbar.
- `nie` sei die überall auf `nil` gesetzte Funktion. O.B.d.A `nie` $\in S$.
- Die Transformation $T(\langle M \rangle) = \langle M^* \rangle$ soll erfüllen

$$\begin{aligned} \langle M \rangle \in \overline{H_\epsilon} \quad &\Leftrightarrow \quad M \text{ hält nicht auf } \epsilon \\ &\overset{!}{\Leftrightarrow} \quad M^* \text{ berechnet } \texttt{nie} \end{aligned}$$

Der Satz von Rice (2/3)

Wenn S nicht-trivial ist, dann ist

$$\mathrm{B}(S) = \{\langle M \rangle \mid \text{die von } M \text{ berechnete Funktion gehört zu } S\}$$

unentscheidbar.

- Wir weisen die Reduktion $\overline{H_\epsilon} \leqslant \mathrm{B}(S)$ nach und $B(S)$ ist deshalb nicht entscheidbar.
- `nie` sei die überall auf `nil` gesetzte Funktion. O.B.d.A `nie` $\in S$.
- Die Transformation $T(\langle M \rangle) = \langle M^* \rangle$ soll erfüllen

$$\begin{aligned} \langle M \rangle \in \overline{H_\epsilon} \quad &\Leftrightarrow \quad M \text{ hält nicht auf } \epsilon \\ &\overset{!}{\Leftrightarrow} \quad M^* \text{ berechnet } \texttt{nie} \overset{!}{\Leftrightarrow} \langle M^* \rangle \in \mathrm{B}(S). \end{aligned}$$

Der Satz von Rice (2/3)

Wenn S nicht-trivial ist, dann ist

$$\mathrm{B}(S) = \{\langle M \rangle \mid \text{die von } M \text{ berechnete Funktion gehört zu } S\}$$

unentscheidbar.

- Wir weisen die Reduktion $\overline{H_\epsilon} \leqslant \mathrm{B}(S)$ nach und $B(S)$ ist deshalb nicht entscheidbar.
- `nie` sei die überall auf `nil` gesetzte Funktion. O.B.d.A `nie` $\in S$.
- Die Transformation $T(\langle M \rangle) = \langle M^* \rangle$ soll erfüllen

$$\begin{aligned} \langle M \rangle \in \overline{H_\epsilon} \quad &\Leftrightarrow \quad M \text{ hält nicht auf } \epsilon \\ &\overset{!}{\Leftrightarrow} \quad M^* \text{ berechnet } \texttt{nie} \overset{!}{\Leftrightarrow} \langle M^* \rangle \in \mathrm{B}(S). \end{aligned}$$

- M^* sollte zuerst M auf ϵ simulieren.

Der Satz von Rice (2/3)

Wenn S nicht-trivial ist, dann ist

$$\mathrm{B}(S) = \{\langle M \rangle \mid \text{die von } M \text{ berechnete Funktion gehört zu } S\}$$

unentscheidbar.

- Wir weisen die Reduktion $\overline{H_\epsilon} \leqslant \mathrm{B}(S)$ nach und $B(S)$ ist deshalb nicht entscheidbar.
- `nie` sei die überall auf `nil` gesetzte Funktion. O.B.d.A `nie` $\in S$.
- Die Transformation $T(\langle M \rangle) = \langle M^* \rangle$ soll erfüllen

$$\begin{aligned} \langle M \rangle \in \overline{H_\epsilon} \quad &\Leftrightarrow \quad M \text{ hält nicht auf } \epsilon \\ &\overset{!}{\Leftrightarrow} \quad M^* \text{ berechnet } \texttt{nie} \overset{!}{\Leftrightarrow} \langle M^* \rangle \in \mathrm{B}(S). \end{aligned}$$

- M^* sollte zuerst M auf ϵ simulieren.
- Wenn M auf ϵ hält, dann sollte M^*

Der Satz von Rice (2/3)

Wenn S nicht-trivial ist, dann ist

$$\mathrm{B}(S) = \{\langle M \rangle \mid \text{die von } M \text{ berechnete Funktion gehört zu } S\}$$

unentscheidbar.

- Wir weisen die Reduktion $\overline{H_\epsilon} \leqslant \mathrm{B}(S)$ nach und $B(S)$ ist deshalb nicht entscheidbar.
- `nie` sei die überall auf `nil` gesetzte Funktion. O.B.d.A `nie` $\in S$.
- Die Transformation $T(\langle M \rangle) = \langle M^* \rangle$ soll erfüllen

$$\begin{aligned} \langle M \rangle \in \overline{H_\epsilon} \quad &\Leftrightarrow \quad M \text{ hält nicht auf } \epsilon \\ &\overset{!}{\Leftrightarrow} \quad M^* \text{ berechnet } \texttt{nie} \overset{!}{\Leftrightarrow} \langle M^* \rangle \in \mathrm{B}(S). \end{aligned}$$

- M^* sollte zuerst M auf ϵ simulieren.
- Wenn M auf ϵ hält, dann sollte M^* eine Funktion in $\overline{S}$ berechnen!

Der Satz von Rice (3/3)

- f sei eine beliebige Funktion, die nicht in S liegt.
  Die Turingmaschine M_0 berechne f.

Der Satz von Rice (3/3)

- f sei eine beliebige Funktion, die nicht in S liegt.

  Die Turingmaschine M_0 berechne f.
- Die Turingmaschine M^*

Der Satz von Rice (3/3)

- f sei eine beliebige Funktion, die nicht in S liegt.

  Die Turingmaschine M_0 berechne f.
- Die Turingmaschine M^*
  - simuliert zuerst M auf dem leeren Wort.

    Wenn M auf ϵ nicht hält, dann berechnet M^* die Funktion `nie` $\in S$.

Der Satz von Rice (3/3)

- f sei eine beliebige Funktion, die nicht in S liegt.

  Die Turingmaschine M_0 berechne f.
- Die Turingmaschine M^*
  - simuliert zuerst M auf dem leeren Wort.

    Wenn M auf ϵ nicht hält, dann berechnet M^* die Funktion `nie` $\in S$.
  - Sollte M auf ϵ halten, dann simuliert M die Maschine M_0 auf der ursprünglichen Eingabe.

Der Satz von Rice (3/3)

- f sei eine beliebige Funktion, die nicht in S liegt.

  Die Turingmaschine M_0 berechne f.
- Die Turingmaschine M^*
  - simuliert zuerst M auf dem leeren Wort.

    Wenn M auf ϵ nicht hält, dann berechnet M^* die Funktion `nie` $\in S$.
  - Sollte M auf ϵ halten, dann simuliert M die Maschine M_0 auf der ursprünglichen Eingabe.

    Wenn M auf ϵ hält, dann berechnet M^* die Funktion $f \notin S$.

Der Satz von Rice (3/3)

- f sei eine beliebige Funktion, die nicht in S liegt.

  Die Turingmaschine M_0 berechne f.
- Die Turingmaschine M^*
  - simuliert zuerst M auf dem leeren Wort.

    Wenn M auf ϵ nicht hält, dann berechnet M^* die Funktion $\texttt{nie} \in S$.
  - Sollte M auf ϵ halten, dann simuliert M die Maschine M_0 auf der ursprünglichen Eingabe.

    Wenn M auf ϵ hält, dann berechnet M^* die Funktion $f \notin S$.

$$\langle M \rangle \in \overline{H_\epsilon} \quad \Leftrightarrow \quad M \text{ hält nicht auf } \epsilon$$

Der Satz von Rice (3/3)

- f sei eine beliebige Funktion, die nicht in S liegt.

  Die Turingmaschine M_0 berechne f.
- Die Turingmaschine M^*
  - simuliert zuerst M auf dem leeren Wort.

    Wenn M auf ϵ nicht hält, dann berechnet M^* die Funktion $\texttt{nie} \in S$.
  - Sollte M auf ϵ halten, dann simuliert M die Maschine M_0 auf der ursprünglichen Eingabe.

    Wenn M auf ϵ hält, dann berechnet M^* die Funktion $f \notin S$.

$$\begin{aligned} \langle M \rangle \in \overline{H_\epsilon} &\Leftrightarrow M \text{ hält nicht auf } \epsilon \\ &\Leftrightarrow M^* \text{ berechnet } \texttt{nie} \end{aligned}$$

Der Satz von Rice (3/3)

- f sei eine beliebige Funktion, die nicht in S liegt.

  Die Turingmaschine M_0 berechne f.
- Die Turingmaschine M^*
  - simuliert zuerst M auf dem leeren Wort.

    Wenn M auf ϵ nicht hält, dann berechnet M^* die Funktion `nie` $\in S$.
  - Sollte M auf ϵ halten, dann simuliert M die Maschine M_0 auf der ursprünglichen Eingabe.

    Wenn M auf ϵ hält, dann berechnet M^* die Funktion $f \notin S$.

$$\begin{aligned} \langle M \rangle \in \overline{H_\epsilon} &\Leftrightarrow M \text{ hält nicht auf } \epsilon \\ &\Leftrightarrow M^* \text{ berechnet } \texttt{nie} \\ &\Leftrightarrow \langle M^* \rangle \in \mathrm{B}(S) \end{aligned}$$

Der Satz von Rice (3/3)

- f sei eine beliebige Funktion, die nicht in S liegt.

  Die Turingmaschine M_0 berechne f.
- Die Turingmaschine M^*
  - simuliert zuerst M auf dem leeren Wort.

    Wenn M auf ϵ nicht hält, dann berechnet M^* die Funktion `nie` $\in S$.
  - Sollte M auf ϵ halten, dann simuliert M die Maschine M_0 auf der ursprünglichen Eingabe.

    Wenn M auf ϵ hält, dann berechnet M^* die Funktion $f \notin S$.

$$\begin{aligned} \langle M \rangle \in \overline{H_\epsilon} &\Leftrightarrow M \text{ hält nicht auf } \epsilon \\ &\Leftrightarrow M^* \text{ berechnet } \texttt{nie} \\ &\Leftrightarrow \langle M^* \rangle \in \mathrm{B}(S) \end{aligned}$$

und das war verlangt: Setze $T(\langle M \rangle) = \langle M^* \rangle$.

Rekursive Aufzählbarkeit

Rekursive Aufzählbarkeit

Ein Entscheidungsproblem L ist genau dann rekursiv aufzählbar (oder auch semi-entscheidbar), wenn es eine Turingmaschine M gibt mit

$$w \in L \Leftrightarrow M \text{ akzeptiert } w.$$

Rekursive Aufzählbarkeit

Ein Entscheidungsproblem L ist genau dann rekursiv aufzählbar (oder auch semi-entscheidbar), wenn es eine Turingmaschine M gibt mit

$$w \in L \Leftrightarrow M \text{ akzeptiert } w.$$

- ACHTUNG: M darf Eingaben $w \notin L$ nicht akzeptieren, muss aber nicht halten!

Rekursive Aufzählbarkeit

Ein Entscheidungsproblem L ist genau dann rekursiv aufzählbar (oder auch semi-entscheidbar), wenn es eine Turingmaschine M gibt mit

$$w \in L \Leftrightarrow M \text{ akzeptiert } w.$$

- ACHTUNG: M darf Eingaben $w \notin L$ nicht akzeptieren, muss aber nicht halten!
- Warum haben wir den Begriff „rekursive Aufzählbarkeit“ gewählt?
  - L ist rekursiv aufzählbar $\Leftrightarrow$ es gibt eine deterministische TM, die genau die Worte $w \in L$ in irgendeiner Reihenfolge, durch das Sonderzeichen # getrennt, auf ein Ausgabeband schreibt.

H ist rekursiv aufzählbar, aber nicht entscheidbar

- Das Halteproblem H ist rekursiv aufzählbar:

H ist rekursiv aufzählbar, aber nicht entscheidbar

- Das Halteproblem H ist rekursiv aufzählbar:
  - Für Eingabe w überprüfe zuerst, ob $w = \langle M \rangle x$.

H ist rekursiv aufzählbar, aber nicht entscheidbar

- Das Halteproblem H ist rekursiv aufzählbar:
  - Für Eingabe w überprüfe zuerst, ob $w = \langle M \rangle x$.
    - ★ Wenn nicht, verwerfe.

H ist rekursiv aufzählbar, aber nicht entscheidbar

- Das Halteproblem H ist rekursiv aufzählbar:
  - Für Eingabe w überprüfe zuerst, ob $w = \langle M \rangle x$.
    - ⋆ Wenn nicht, verwerfe.
  - Ansonsten simuliere M auf Eingabe x. Sollte M auf x halten, so akzeptiere.

H ist rekursiv aufzählbar, aber nicht entscheidbar

- Das Halteproblem H ist rekursiv aufzählbar:
  - Für Eingabe w überprüfe zuerst, ob $w = \langle M \rangle x$.
    - ⋆ Wenn nicht, verwerfe.
  - Ansonsten simuliere M auf Eingabe x. Sollte M auf x halten, so akzeptiere.
  - Die Simulation akzeptiert jede Eingabe $\langle M \rangle x \in H$.
    Die verbleibenden Eingaben verwirft M oder M hält nicht.

H ist rekursiv aufzählbar, aber nicht entscheidbar

- Das Halteproblem H ist rekursiv aufzählbar:
  - Für Eingabe w überprüfe zuerst, ob $w = \langle M \rangle x$.
    - ⋆ Wenn nicht, verwerfe.
  - Ansonsten simuliere M auf Eingabe x. Sollte M auf x halten, so akzeptiere.
  - Die Simulation akzeptiert jede Eingabe $\langle M \rangle x \in H$.
    Die verbleibenden Eingaben verwirft M oder M hält nicht.
- Mit derselben Argumentation: H_ϵ und U sind rekursiv aufzählbar, aber nicht entscheidbar.

Rekursive Aufzählbarkeit und Entscheidbarkeit

- Jedes entscheidbare Problem ist natürlich rekursiv aufzählbar.

Rekursive Aufzählbarkeit und Entscheidbarkeit

- Jedes entscheidbare Problem ist natürlich rekursiv aufzählbar.
- Die Klasse der rekursiv aufzählbaren Probleme ist größer und enthält nicht-entscheidbare Probleme wie H, H_ϵ und U.

Rekursive Aufzählbarkeit und Entscheidbarkeit

- Jedes entscheidbare Problem ist natürlich rekursiv aufzählbar.
- Die Klasse der rekursiv aufzählbaren Probleme ist größer und enthält nicht-entscheidbare Probleme wie H, H_ϵ und U.

! Wenn L und $\overline{L}$ rekursiv aufzählbar sind, dann ist L entscheidbar.

Rekursive Aufzählbarkeit und Entscheidbarkeit

- Jedes entscheidbare Problem ist natürlich rekursiv aufzählbar.
- Die Klasse der rekursiv aufzählbaren Probleme ist größer und enthält nicht-entscheidbare Probleme wie H, H_ϵ und U.

! Wenn L und $\overline{L}$ rekursiv aufzählbar sind, dann ist L entscheidbar.

- Die Turingmaschine M_1 löse das Problem L und M_2 löse das Komplementproblem $\overline{L}$.

Rekursive Aufzählbarkeit und Entscheidbarkeit

- Jedes entscheidbare Problem ist natürlich rekursiv aufzählbar.
- Die Klasse der rekursiv aufzählbaren Probleme ist größer und enthält nicht-entscheidbare Probleme wie H, H_ϵ und U.

! Wenn L und $\overline{L}$ rekursiv aufzählbar sind, dann ist L entscheidbar.

- Die Turingmaschine M_1 löse das Problem L und M_2 löse das Komplementproblem $\overline{L}$.
- Wir bauen eine Turingmaschine M für L, die die Maschinen M_1 und M_2 verzahnt simuliert:

Rekursive Aufzählbarkeit und Entscheidbarkeit

- Jedes entscheidbare Problem ist natürlich rekursiv aufzählbar.
- Die Klasse der rekursiv aufzählbaren Probleme ist größer und enthält nicht-entscheidbare Probleme wie H, H_ϵ und U.

! Wenn L und $\overline{L}$ rekursiv aufzählbar sind, dann ist L entscheidbar.

- Die Turingmaschine M_1 löse das Problem L und M_2 löse das Komplementproblem $\overline{L}$.
- Wir bauen eine Turingmaschine M für L, die die Maschinen M_1 und M_2 verzahnt simuliert:
  - Auf jeden Schritt von M_1 folgt ein Schritt von M_2 und umgekehrt.

Rekursive Aufzählbarkeit und Entscheidbarkeit

- Jedes entscheidbare Problem ist natürlich rekursiv aufzählbar.
- Die Klasse der rekursiv aufzählbaren Probleme ist größer und enthält nicht-entscheidbare Probleme wie H, H_ϵ und U.

! Wenn L und $\overline{L}$ rekursiv aufzählbar sind, dann ist L entscheidbar.

- Die Turingmaschine M_1 löse das Problem L und M_2 löse das Komplementproblem $\overline{L}$.
- Wir bauen eine Turingmaschine M für L, die die Maschinen M_1 und M_2 verzahnt simuliert:
  - Auf jeden Schritt von M_1 folgt ein Schritt von M_2 und umgekehrt.
  - Eine der beiden Maschinen wird halten und akzeptieren!
    - ★ Wenn M_1 hält und akzeptiert, dann auch M.
    - ★ Wenn M_2 hält und akzeptiert, dann hält M und verwirft.

Rekursive Aufzählbarkeit und Entscheidbarkeit

- Jedes entscheidbare Problem ist natürlich rekursiv aufzählbar.
- Die Klasse der rekursiv aufzählbaren Probleme ist größer und enthält nicht-entscheidbare Probleme wie H, H_ϵ und U.

! Wenn L und $\overline{L}$ rekursiv aufzählbar sind, dann ist L entscheidbar.

- Die Turingmaschine M_1 löse das Problem L und M_2 löse das Komplementproblem $\overline{L}$.
- Wir bauen eine Turingmaschine M für L, die die Maschinen M_1 und M_2 verzahnt simuliert:
  - Auf jeden Schritt von M_1 folgt ein Schritt von M_2 und umgekehrt.
  - Eine der beiden Maschinen wird halten und akzeptieren!
    - ⋆ Wenn M_1 hält und akzeptiert, dann auch M.
    - ⋆ Wenn M_2 hält und akzeptiert, dann hält M und verwirft.
- M ist eine stets haltende Turingmaschine, die das Problem L löst.

Rekursive Aufzählbarkeit und Entscheidbarkeit

- Jedes entscheidbare Problem ist natürlich rekursiv aufzählbar.
- Die Klasse der rekursiv aufzählbaren Probleme ist größer und enthält nicht-entscheidbare Probleme wie H, H_ϵ und U.

! Wenn L und $\overline{L}$ rekursiv aufzählbar sind, dann ist L entscheidbar.

- Die Turingmaschine M_1 löse das Problem L und M_2 löse das Komplementproblem $\overline{L}$.
- Wir bauen eine Turingmaschine M für L, die die Maschinen M_1 und M_2 verzahnt simuliert:
  - Auf jeden Schritt von M_1 folgt ein Schritt von M_2 und umgekehrt.
  - Eine der beiden Maschinen wird halten und akzeptieren!
    - ★ Wenn M_1 hält und akzeptiert, dann auch M.
    - ★ Wenn M_2 hält und akzeptiert, dann hält M und verwirft.
- M ist eine stets haltende Turingmaschine, die das Problem L löst.

$\overline{H}$, $\overline{H_\epsilon}$ und $\overline{U}$ sind nicht rekursiv aufzählbar!

Rekursive Aufzählbarkeit und Grammatiken

Sei G eine Grammatik

$L(G)$ ist die Menge aller von G erzeugten Worte.

Dann ist $L(G)$ rekursiv aufzählbar.

Rekursive Aufzählbarkeit und Grammatiken

Sei G eine Grammatik

$L(G)$ ist die Menge aller von G erzeugten Worte.

Dann ist $L(G)$ rekursiv aufzählbar.

- Angenommen, wir müssen entscheiden, ob ein Wort w von G erzeugt wird.

Rekursive Aufzählbarkeit und Grammatiken

Sei G eine Grammatik

$L(G)$ ist die Menge aller von G erzeugten Worte.

Dann ist $L(G)$ rekursiv aufzählbar.

- Angenommen, wir müssen entscheiden, ob ein Wort w von G erzeugt wird.
- Zähle alle möglichen Folgen von Produktionen auf.

Rekursive Aufzählbarkeit und Grammatiken

Sei G eine Grammatik

$L(G)$ ist die Menge aller von G erzeugten Worte.

Dann ist $L(G)$ rekursiv aufzählbar.

- Angenommen, wir müssen entscheiden, ob ein Wort w von G erzeugt wird.
- Zähle alle möglichen Folgen von Produktionen auf.
  - Wenn $w \in L(G)$, dann werden wir eine Folge von Produktionen finden, die w erzeugt.

Rekursive Aufzählbarkeit und Grammatiken

Sei G eine Grammatik

$L(G)$ ist die Menge aller von G erzeugten Worte.

Dann ist $L(G)$ rekursiv aufzählbar.

- Angenommen, wir müssen entscheiden, ob ein Wort w von G erzeugt wird.
- Zähle alle möglichen Folgen von Produktionen auf.
  - Wenn $w \in L(G)$, dann werden wir eine Folge von Produktionen finden, die w erzeugt.
  - Wenn $w \notin L(G)$, wird unsere Berechnung nicht halten, aber dies ist auch nicht erforderlich.

Rekursive Aufzählbarkeit und logische Theorien

Sei Ax ein Axiomensystem.

$Th(Ax)$ ist die Menge aller aus Ax ableitbaren Aussagen.

Dann ist $Th(Ax)$ rekursiv aufzählbar.

Rekursive Aufzählbarkeit und logische Theorien

Sei Ax ein Axiomensystem.

$Th(Ax)$ ist die Menge aller aus Ax ableitbaren Aussagen.

Dann ist $Th(Ax)$ rekursiv aufzählbar.

- Angenommen, wir müssen entscheiden, ob eine Aussage α aus Ax ableitbar ist.

Rekursive Aufzählbarkeit und logische Theorien

Sei Ax ein Axiomensystem.

$Th(Ax)$ ist die Menge aller aus Ax ableitbaren Aussagen.

Dann ist $Th(Ax)$ rekursiv aufzählbar.

- Angenommen, wir müssen entscheiden, ob eine Aussage α aus Ax ableitbar ist.
- Zähle alle möglichen Ableitungen aus den Axiomen in Ax auf.

Rekursive Aufzählbarkeit und logische Theorien

Sei Ax ein Axiomensystem.

$Th(Ax)$ ist die Menge aller aus Ax ableitbaren Aussagen.

Dann ist $Th(Ax)$ rekursiv aufzählbar.

- Angenommen, wir müssen entscheiden, ob eine Aussage α aus Ax ableitbar ist.
- Zähle alle möglichen Ableitungen aus den Axiomen in Ax auf.
  - Wenn $\alpha \in Th(Ax)$, dann werden wir eine Ableitung finden.

Rekursive Aufzählbarkeit und logische Theorien

Sei Ax ein Axiomensystem.

$Th(Ax)$ ist die Menge aller aus Ax ableitbaren Aussagen.

Dann ist $Th(Ax)$ rekursiv aufzählbar.

- Angenommen, wir müssen entscheiden, ob eine Aussage α aus Ax ableitbar ist.
- Zähle alle möglichen Ableitungen aus den Axiomen in Ax auf.
  - Wenn $\alpha \in Th(Ax)$, dann werden wir eine Ableitung finden.
  - Wenn $\alpha \notin Th(Ax)$, wird unsere Berechnung nicht halten, aber was soll's?

Der Gödelsche Unvollständigkeitssatz

Das Hilbertsche Programm

Im Jahr 1903 zeigt Bertrand Russell, dass die naive Mengenlehre nicht widerspruchsfrei ist.

Das Hilbertsche Programm

Im Jahr 1903 zeigt Bertrand Russell, dass die naive Mengenlehre nicht widerspruchsfrei ist.
Als Reaktion darauf entwickelt David Hilbert in den 1920'er Jahren das Hilbertsche Programm:

Das Hilbertsche Programm

Im Jahr 1903 zeigt Bertrand Russell, dass die naive Mengenlehre nicht widerspruchsfrei ist.
Als Reaktion darauf entwickelt David Hilbert in den 1920'er Jahren das Hilbertsche Programm:

- Finde ein Axiomensystem mit unmittelbar einsichtigen Axiomen, um die Aussagen der Mathematik zweifelsfrei nachzuweisen.

Das Hilbertsche Programm

Im Jahr 1903 zeigt Bertrand Russell, dass die naive Mengenlehre nicht widerspruchsfrei ist.
Als Reaktion darauf entwickelt David Hilbert in den 1920'er Jahren das Hilbertsche Programm:

- Finde ein Axiomensystem mit unmittelbar einsichtigen Axiomen, um die Aussagen der Mathematik zweifelsfrei nachzuweisen.
- Das Axiomensystem und seine Schlussregeln müssen mächtig genug sein, um alle wahren Aussagen abzuleiten.

Das Hilbertsche Programm

Im Jahr 1903 zeigt Bertrand Russell, dass die naive Mengenlehre nicht widerspruchsfrei ist.
Als Reaktion darauf entwickelt David Hilbert in den 1920'er Jahren das Hilbertsche Programm:

- Finde ein Axiomensystem mit unmittelbar einsichtigen Axiomen, um die Aussagen der Mathematik zweifelsfrei nachzuweisen.
- Das Axiomensystem und seine Schlussregeln müssen mächtig genug sein, um alle wahren Aussagen abzuleiten.

Kurt Gödel zeigt mit seinem Unvollständigkeitssatz im Jahr 1932, dass das Hilbertsche Programm nicht wie ursprünglich gedacht durchgeführt werden kann.

Die Peano-Arithmetik

Die Peano-Arithmetik ist ein in der Prädikatenlogik der ersten Stufe formalisiertes Axiomensystem für die natürlichen Zahlen:

Die Peano-Arithmetik

Die Peano-Arithmetik ist ein in der Prädikatenlogik der ersten Stufe formalisiertes Axiomensystem für die natürlichen Zahlen:

- Die Axiome der Peano-Arithmetik formulieren die Eigenschaften der Addition

  $x + 0 = x$,

Die Peano-Arithmetik

Die Peano-Arithmetik ist ein in der Prädikatenlogik der ersten Stufe formalisiertes Axiomensystem für die natürlichen Zahlen:

- Die Axiome der Peano-Arithmetik formulieren die Eigenschaften der Addition

  $x + 0 = x$, $x + (y + 1) = (x + y) + 1$,

Die Peano-Arithmetik

Die Peano-Arithmetik ist ein in der Prädikatenlogik der ersten Stufe formalisiertes Axiomensystem für die natürlichen Zahlen:

- Die Axiome der Peano-Arithmetik formulieren die Eigenschaften der Addition

  $$x + 0 = x,\ x + (y + 1) = (x + y) + 1,$$

- der Multiplikation

  $$x \cdot 0 = 0,$$

Die Peano-Arithmetik

Die Peano-Arithmetik ist ein in der Prädikatenlogik der ersten Stufe formalisiertes Axiomensystem für die natürlichen Zahlen:

- Die Axiome der Peano-Arithmetik formulieren die Eigenschaften der Addition

  $$x + 0 = x,\ x + (y + 1) = (x + y) + 1,$$

- der Multiplikation

  $$x \cdot 0 = 0,\ x \cdot (y + 1) = x \cdot y + x,$$

Die Peano-Arithmetik

Die Peano-Arithmetik ist ein in der Prädikatenlogik der ersten Stufe formalisiertes Axiomensystem für die natürlichen Zahlen:

- Die Axiome der Peano-Arithmetik formulieren die Eigenschaften der Addition

  $x + 0 = x$, $x + (y + 1) = (x + y) + 1$,
- der Multiplikation

  $x \cdot 0 = 0$, $x \cdot (y + 1) = x \cdot y + x$,
- sowie die Induktionseigenschaft:

  Für jede Formel $\phi(x)$ ist die Formel

  $$(\phi(0) \wedge \forall x(\phi(x) \rightarrow \phi(x + 1))) \rightarrow \forall x \phi(x)$$

  ein Axiom.

Die Peano-Arithmetik

Die Peano-Arithmetik ist ein in der Prädikatenlogik der ersten Stufe formalisiertes Axiomensystem für die natürlichen Zahlen:

- Die Axiome der Peano-Arithmetik formulieren die Eigenschaften der Addition

  $x + 0 = x$, $x + (y + 1) = (x + y) + 1$,
- der Multiplikation

  $x \cdot 0 = 0$, $x \cdot (y + 1) = x \cdot y + x$,
- sowie die Induktionseigenschaft:

  Für jede Formel $\phi(x)$ ist die Formel

  $$(\phi(0) \wedge \forall x(\phi(x) \rightarrow \phi(x + 1))) \rightarrow \forall x \phi(x)$$

  ein Axiom.

Lassen sich alle für die natürlichen Zahlen wahren Aussagen auch formal ableiten?

Man kann zeigen: Für jedes rekursiv aufzählbare Problem L über den natürlichen Zahlen (bzw. über $\{0,1\}^*$) gibt es eine Formel φ_L der Peano-Arithmetik mit

$$x \in L \Leftrightarrow \varphi_L(x) \text{ ist wahr}$$

Man kann zeigen: Für jedes rekursiv aufzählbare Problem L über den natürlichen Zahlen (bzw. über $\{0,1\}^*$) gibt es eine Formel φ_L der Peano-Arithmetik mit

$$x \in L \Leftrightarrow \varphi_L(x) \text{ ist wahr}$$

- Wenn Wahrheit und Ableitbarkeit übereinstimmen, dann folgt also

$$x \in L \Leftrightarrow \varphi_L(x) \text{ ist in der Peano Arithmetik beweisbar}$$

Man kann zeigen: Für jedes rekursiv aufzählbare Problem L über den natürlichen Zahlen (bzw. über $\{0,1\}^*$) gibt es eine Formel φ_L der Peano-Arithmetik mit

$$x \in L \Leftrightarrow \varphi_L(x) \text{ ist wahr}$$

- Wenn Wahrheit und Ableitbarkeit übereinstimmen, dann folgt also

$$x \in L \Leftrightarrow \varphi_L(x) \text{ ist in der Peano Arithmetik beweisbar}$$

und ebenso natürlich

$$x \notin L \quad \Leftrightarrow \quad \neg\varphi_L(x) \text{ ist wahr.}$$

Man kann zeigen: Für jedes rekursiv aufzählbare Problem L über den natürlichen Zahlen (bzw. über $\{0,1\}^*$) gibt es eine Formel φ_L der Peano-Arithmetik mit

$$x \in L \Leftrightarrow \varphi_L(x) \text{ ist wahr}$$

- Wenn Wahrheit und Ableitbarkeit übereinstimmen, dann folgt also

$$x \in L \Leftrightarrow \varphi_L(x) \text{ ist in der Peano Arithmetik beweisbar}$$

und ebenso natürlich

$$\begin{aligned} x \notin L &\Leftrightarrow \neg\varphi_L(x) \text{ ist wahr.} \\ &\Leftrightarrow (\neg\varphi_L(x)) \text{ ist beweisbar.} \end{aligned}$$

Man kann zeigen: Für jedes rekursiv aufzählbare Problem L über den natürlichen Zahlen (bzw. über $\{0,1\}^*$) gibt es eine Formel φ_L der Peano-Arithmetik mit

$$x \in L \Leftrightarrow \varphi_L(x) \text{ ist wahr}$$

- Wenn Wahrheit und Ableitbarkeit übereinstimmen, dann folgt also

$$x \in L \Leftrightarrow \varphi_L(x) \text{ ist in der Peano Arithmetik beweisbar}$$

und ebenso natürlich

$$\begin{aligned} x \notin L &\Leftrightarrow \neg\varphi_L(x) \text{ ist wahr.} \\ &\Leftrightarrow (\neg\varphi_L(x)) \text{ ist beweisbar.} \end{aligned}$$

- Also ist $\bar{L} = \{x | (\neg\varphi_L(x)) \text{ ist beweisbar } \}$.

Der Gödelsche Unvollständigkeitssatz (1/2)

- Aber $\bar{L} = \{x | (\neg \varphi_L(x)) \text{ ist beweisbar } \}$ ist dann auch rekursiv aufzählbar.

Der Gödelsche Unvollständigkeitssatz (1/2)

- Aber $\bar{L} = \{x | (\neg\varphi_L(x)) \text{ ist beweisbar } \}$ ist dann auch rekursiv aufzählbar.

  Um zu prüfen, ob eine Formel φ ableitbar ist, zähle alle möglichen Beweise auf und akzeptiere, wenn ein Beweis für φ gefunden wird.

Der Gödelsche Unvollständigkeitssatz (1/2)

- Aber $\bar{L} = \{x | (\neg\varphi_L(x)) \text{ ist beweisbar } \}$ ist dann auch rekursiv aufzählbar.

  Um zu prüfen, ob eine Formel φ ableitbar ist, zähle alle möglichen Beweise auf und akzeptiere, wenn ein Beweis für φ gefunden wird.
- L war aber eine beliebiges rekursiv aufzählbares Problem und somit ist jedes rekursiv aufzählbare Problem auch entscheidbar.

Der Gödelsche Unvollständigkeitssatz (1/2)

- Aber $\bar{L} = \{x | (\neg\varphi_L(x)) \text{ ist beweisbar } \}$ ist dann auch rekursiv aufzählbar.

  Um zu prüfen, ob eine Formel φ ableitbar ist, zähle alle möglichen Beweise auf und akzeptiere, wenn ein Beweis für φ gefunden wird.
- L war aber eine beliebiges rekursiv aufzählbares Problem und somit ist jedes rekursiv aufzählbare Problem auch entscheidbar.

  Blanker Unsinn: Beweisbarkeit ist also schwächer als Wahrheit!

Der Gödelsche Unvollständigkeitssatz (1/2)

- Aber $\bar{L} = \{x | (\neg\varphi_L(x)) \text{ ist beweisbar } \}$ ist dann auch rekursiv aufzählbar.

  Um zu prüfen, ob eine Formel φ ableitbar ist, zähle alle möglichen Beweise auf und akzeptiere, wenn ein Beweis für φ gefunden wird.
- L war aber eine beliebiges rekursiv aufzählbares Problem und somit ist jedes rekursiv aufzählbare Problem auch entscheidbar.

  Blanker Unsinn: Beweisbarkeit ist also schwächer als Wahrheit!

Und wenn wir das Axiomensystem um weitere wahre Aussagen erweitern?

Der Gödelsche Unvollständigkeitssatz (2/2)

Wir fügen weitere, möglicherweise sogar unendlich viele wahre Aussagen als Axiome hinzu und erhalten ein neues Axiomensystem P.

Der Gödelsche Unvollständigkeitssatz (2/2)

Wir fügen weitere, möglicherweise sogar unendlich viele wahre Aussagen als Axiome hinzu und erhalten ein neues Axiomensystem P.

- Wenn P nicht rekursiv aufzählbar ist, dann können wir noch nicht einmal verifizieren, dass ein Axiom zu P gehört.

Der Gödelsche Unvollständigkeitssatz (2/2)

Wir fügen weitere, möglicherweise sogar unendlich viele wahre Aussagen als Axiome hinzu und erhalten ein neues Axiomensystem P.

- Wenn P nicht rekursiv aufzählbar ist, dann können wir noch nicht einmal verifizieren, dass ein Axiom zu P gehört.
- Wenn das Axiomensystem P aber rekursiv aufzählbar ist, und wenn Wahrheit und Beweisbarkeit diesmal übereinstimmen, dann

Der Gödelsche Unvollständigkeitssatz (2/2)

Wir fügen weitere, möglicherweise sogar unendlich viele wahre Aussagen als Axiome hinzu und erhalten ein neues Axiomensystem P.

- Wenn P nicht rekursiv aufzählbar ist, dann können wir noch nicht einmal verifizieren, dass ein Axiom zu P gehört.
- Wenn das Axiomensystem P aber rekursiv aufzählbar ist, und wenn Wahrheit und Beweisbarkeit diesmal übereinstimmen, dann bleibt die Menge $\overline{L}$ für jede rekursiv aufzählbare Menge L rekursiv aufzählbar!

Der Gödelsche Unvollständigkeitssatz (2/2)

Wir fügen weitere, möglicherweise sogar unendlich viele wahre Aussagen als Axiome hinzu und erhalten ein neues Axiomensystem P.

- Wenn P nicht rekursiv aufzählbar ist, dann können wir noch nicht einmal verifizieren, dass ein Axiom zu P gehört.
- Wenn das Axiomensystem P aber rekursiv aufzählbar ist, und wenn Wahrheit und Beweisbarkeit diesmal übereinstimmen, dann bleibt die Menge $\overline{L}$ für jede rekursiv aufzählbare Menge L rekursiv aufzählbar!

Sei P eine rekursiv aufzählbare Menge von wahren Formeln der Zahlentheorie, wobei P die Axiome der Peano-Arithmetik enthalte.
Dann gibt es eine wahre Formel, die nicht aus P ableitbar ist.

Und die Konsequenz für die Informatik

Komplexe Realitäten lassen sich nicht formal beschreiben!

Zusammenfassung (1/2)

- Wir haben
  - die Berechenbarkeit von Funktionen und
  - die Entscheidbarkeit von Problemen

  definiert.

Zusammenfassung (1/2)

- Wir haben
  - die Berechenbarkeit von Funktionen und
  - die Entscheidbarkeit von Problemen

  definiert.
- Wenn die Church-Turing These richtig ist, dann gelten diese Begriffe auch für alle zukünftigen Technologien!

Zusammenfassung (1/2)

- Wir haben
  - die Berechenbarkeit von Funktionen und
  - die Entscheidbarkeit von Problemen

  definiert.
- Wenn die Church-Turing These richtig ist, dann gelten diese Begriffe auch für alle zukünftigen Technologien!
- Die Diagonalsprache $D = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle \text{ nicht } \}$ war unser erstes nicht entscheidbares Problem, das wir mit der Methode der Diagonaliserung untersucht haben.

Zusammenfassung (1/2)

- Wir haben
  - die Berechenbarkeit von Funktionen und
  - die Entscheidbarkeit von Problemen

  definiert.
- Wenn die Church-Turing These richtig ist, dann gelten diese Begriffe auch für alle zukünftigen Technologien!
- Die Diagonalsprache $D = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle \text{ nicht } \}$ war unser erstes nicht entscheidbares Problem, das wir mit der Methode der Diagonaliserung untersucht haben.
- Reduktionen liefern weitere unentscheidbare Probleme:

Zusammenfassung (1/2)

- Wir haben
  - die Berechenbarkeit von Funktionen und
  - die Entscheidbarkeit von Problemen

  definiert.
- Wenn die Church-Turing These richtig ist, dann gelten diese Begriffe auch für alle zukünftigen Technologien!
- Die Diagonalsprache $D = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle \text{ nicht } \}$ war unser erstes nicht entscheidbares Problem, das wir mit der Methode der Diagonaliserung untersucht haben.
- Reduktionen liefern weitere unentscheidbare Probleme:
  - Die universelle Sprache $U = \{\langle M \rangle x \mid M \text{ akzeptiert } x\}$,

Zusammenfassung (1/2)

- Wir haben
  - die Berechenbarkeit von Funktionen und
  - die Entscheidbarkeit von Problemen

  definiert.
- Wenn die Church-Turing These richtig ist, dann gelten diese Begriffe auch für alle zukünftigen Technologien!
- Die Diagonalsprache $D = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle \text{ nicht } \}$ war unser erstes nicht entscheidbares Problem, das wir mit der Methode der Diagonaliserung untersucht haben.
- Reduktionen liefern weitere unentscheidbare Probleme:
  - Die universelle Sprache $U = \{\langle M \rangle x \mid M \text{ akzeptiert } x\}$,
  - das Halteproblem $H = \{\langle M \rangle x \mid M \text{ hält auf } x\}$

Zusammenfassung (1/2)

- Wir haben
  - die Berechenbarkeit von Funktionen und
  - die Entscheidbarkeit von Problemen

  definiert.
- Wenn die Church-Turing These richtig ist, dann gelten diese Begriffe auch für alle zukünftigen Technologien!
- Die Diagonalsprache $D = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle \text{ nicht } \}$ war unser erstes nicht entscheidbares Problem, das wir mit der Methode der Diagonaliserung untersucht haben.
- Reduktionen liefern weitere unentscheidbare Probleme:
  - Die universelle Sprache $U = \{\langle M \rangle x \mid M \text{ akzeptiert } x\}$,
  - das Halteproblem $H = \{\langle M \rangle x \mid M \text{ hält auf } x\}$ und
  - das spezielle Halteproblem $H_\epsilon = \{\langle M \rangle \mid M \text{ hält auf } \epsilon\}$.

Zusammenfassung (1/2)

- Wir haben
  - die Berechenbarkeit von Funktionen und
  - die Entscheidbarkeit von Problemen

  definiert.
- Wenn die Church-Turing These richtig ist, dann gelten diese Begriffe auch für alle zukünftigen Technologien!
- Die Diagonalsprache $D = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle \text{ nicht } \}$ war unser erstes nicht entscheidbares Problem, das wir mit der Methode der Diagonaliserung untersucht haben.
- Reduktionen liefern weitere unentscheidbare Probleme:
  - Die universelle Sprache $U = \{\langle M \rangle x \mid M \text{ akzeptiert } x\}$,
  - das Halteproblem $H = \{\langle M \rangle x \mid M \text{ hält auf } x\}$ und
  - das spezielle Halteproblem $H_\epsilon = \{\langle M \rangle \mid M \text{ hält auf } \epsilon\}$.
- Auch der Satz von Rice liefert weitere unentscheidbare Probleme.

Zusammenfassung (2/2)

Wir haben den Begriff der rekursiven Aufzählbarkeit eingeführt.

- Beispiele rekursiv aufzählbarer Probleme sind U, H und H_ϵ.

Zusammenfassung (2/2)

Wir haben den Begriff der rekursiven Aufzählbarkeit eingeführt.

- Beispiele rekursiv aufzählbarer Probleme sind U, H und H_ϵ.
- Die Menge $L(G)$ aller von der Grammatik G erzeugten Worte ist für jede Grammatik G rekursiv aufzählbar.

Zusammenfassung (2/2)

Wir haben den Begriff der rekursiven Aufzählbarkeit eingeführt.

- Beispiele rekursiv aufzählbarer Probleme sind U, H und H_ϵ.
- Die Menge $L(G)$ aller von der Grammatik G erzeugten Worte ist für jede Grammatik G rekursiv aufzählbar.
- Für ein rekursiv aufzählbares Axiomensystem ist die Menge aller aus dem Axiomensystem ableitbaren Formeln rekursiv aufzählbar.

Zusammenfassung (2/2)

Wir haben den Begriff der rekursiven Aufzählbarkeit eingeführt.

- Beispiele rekursiv aufzählbarer Probleme sind U, H und H_ϵ.
- Die Menge $L(G)$ aller von der Grammatik G erzeugten Worte ist für jede Grammatik G rekursiv aufzählbar.
- Für ein rekursiv aufzählbares Axiomensystem ist die Menge aller aus dem Axiomensystem ableitbaren Formeln rekursiv aufzählbar.
- Wenn ein Problem L und sein Komplement rekursiv aufzählbar sind, dann ist L entscheidbar.

  Direkte oder indirekte Konsequenzen sind:

Zusammenfassung (2/2)

Wir haben den Begriff der rekursiven Aufzählbarkeit eingeführt.

- Beispiele rekursiv aufzählbarer Probleme sind U, H und H_ϵ.
- Die Menge $L(G)$ aller von der Grammatik G erzeugten Worte ist für jede Grammatik G rekursiv aufzählbar.
- Für ein rekursiv aufzählbares Axiomensystem ist die Menge aller aus dem Axiomensystem ableitbaren Formeln rekursiv aufzählbar.
- Wenn ein Problem L und sein Komplement rekursiv aufzählbar sind, dann ist L entscheidbar.

  Direkte oder indirekte Konsequenzen sind:

  - ⋆ Die Komplemente von U, H und H_ϵ sind nicht rekursiv aufzählbar.

Zusammenfassung (2/2)

Wir haben den Begriff der rekursiven Aufzählbarkeit eingeführt.

- Beispiele rekursiv aufzählbarer Probleme sind U, H und H_ϵ.
- Die Menge $L(G)$ aller von der Grammatik G erzeugten Worte ist für jede Grammatik G rekursiv aufzählbar.
- Für ein rekursiv aufzählbares Axiomensystem ist die Menge aller aus dem Axiomensystem ableitbaren Formeln rekursiv aufzählbar.
- Wenn ein Problem L und sein Komplement rekursiv aufzählbar sind, dann ist L entscheidbar.

  Direkte oder indirekte Konsequenzen sind:

  - ⋆ Die Komplemente von U, H und H_ϵ sind nicht rekursiv aufzählbar.
  - ⋆ Der Gödelsche Unvollständigkeitssatz.