universität
innsbruck

Diskrete Mathematik

Ralph Bottesch
Burak Ekici
Johannes Koch
Georg Moser
David Obwaller
Vincent van Oostrom
Oleksandra Panasiuk

cbr.uibk.ac.at

Zusammenfassung der letzten LVA

Satz

jede rekursive Menge ist rekursiv aufzählbar, aber nicht jede rekursiv aufzählbare Menge ist rekursiv

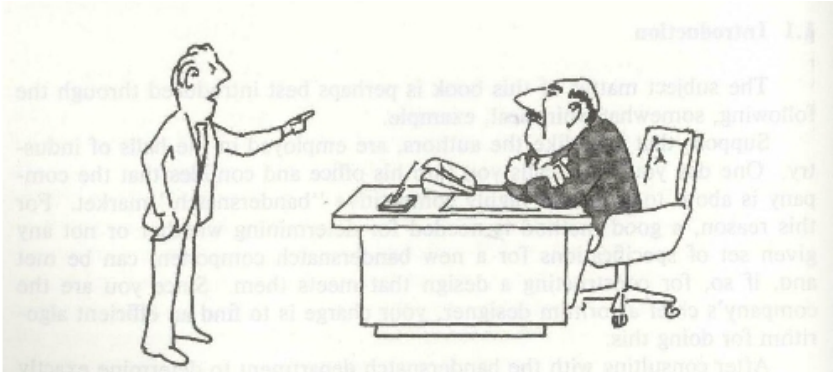
Satz

HP *ist nicht rekursiv, aber rekursiv aufzählbar*

Folgerung

Die Menge $\sim$HP *ist nicht rekursiv aufzählbar*

Berechenbarkeitstheorie, graphisch



"I can't find an efficient algorithm, because no such algorithm is possible!"

Inhalte der Lehrveranstaltung (cont’d)

Reguläre Sprachen

deterministische Automaten, nichtdeterministische Automaten, endliche Automaten mit Epsilon-Übergängen, reguläre Ausdrücke, Abgeschossenheit, Schleifenlemma

Berechenbarkeitstheorie

deterministische TM, nichtdeterministische TM, universelle TMs, Äquivalenzen

Komplexitätstheorie

Grundlagen, die Klassen P und NP, polynomielle Reduktion, logspace Reduktion, die Klassen, NLOGSPACE und PSPACE

Inhalte der Lehrveranstaltung (cont'd)

Reguläre Sprachen

deterministische Automaten, nichtdeterministische Automaten, endliche Automaten mit Epsilon-Übergängen, reguläre Ausdrücke, Abgeschossenheit, Schleifenlemma

Berechenbarkeitstheorie

deterministische TM, nichtdeterministische TM, universelle TMs, Äquivalenzen

Komplexitätstheorie

Grundlagen, die Klassen P und NP, polynomielle Reduktion, logspace Reduktion, die Klassen, NLOGSPACE und PSPACE

Reduktionen

Definition

1. $\exists$ totale DTM T mit Eingabealphabet Σ
2. bei Eingabe $x \in \Sigma^*$, schreibt T $f(x)$ auf das (erste) Band

dann heißt $f\colon \Sigma^* \to \Sigma^*$ berechenbar

Reduktionen

Definition

1. $\exists$ totale DTM T mit Eingabealphabet Σ
2. bei Eingabe $x \in \Sigma^*$, schreibt T $f(x)$ auf das (erste) Band

dann heißt $f\colon \Sigma^* \to \Sigma^*$ berechenbar

Definition

1. $\exists\, R\colon \Sigma^* \to \Sigma^*$
2. R berechenbar
3. für $L \subseteq \Sigma^*$, $M \subseteq \Sigma^*$ gilt $x \in L \Leftrightarrow R(x) \in M$

dann ist L auf M reduzierbar; kurz: $L \leqslant_T M$

Reduktion (schematisch)

angenommen

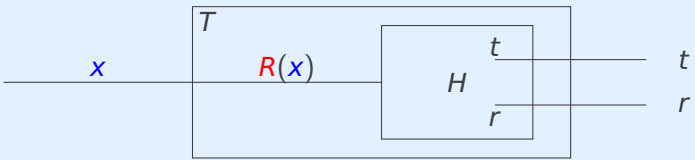
- L, M Sprachen über Σ
- $L \leqslant_T M$ mit $R\colon \Sigma^* \to \Sigma^*$
- die Reduktion R wird von TM T berechnet

Reduktion (schematisch)

angenommen

- L, M Sprachen über Σ
- $L \leqslant_T M$ mit $R\colon \Sigma^* \to \Sigma^*$
- die Reduktion R wird von TM T berechnet

$$x \in L \Leftrightarrow R(x) \in M$$

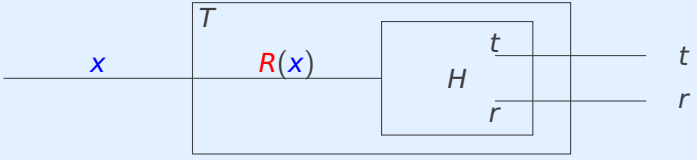


Reduktion (schematisch)

angenommen

- L, M Sprachen über Σ
- $L \leqslant_T M$ mit $R\colon \Sigma^* \to \Sigma^*$
- die Reduktion R wird von TM T berechnet

$$x \in L \Leftrightarrow R(x) \in M$$



Lemma

wenn $L \leqslant_T M$ und M rekursiv, dann ist L rekursiv

Satz

MP *ist nicht rekursiv, aber rekursiv aufzählbar*

Satz

MP *ist nicht rekursiv, aber rekursiv aufzählbar*

Beweisskizze.

1. um zu zeigen, dass MP rekursiv aufzählbar, definiere UTM U, die bei Eingabe $M\#x$, TM M auf x simuliert

Satz

MP *ist nicht rekursiv, aber rekursiv aufzählbar*

Beweisskizze.

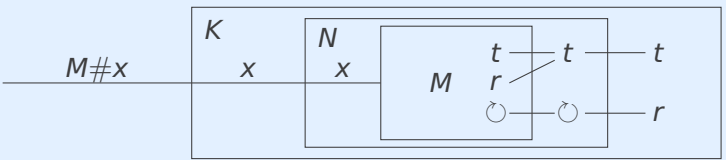
1. um zu zeigen, dass MP rekursiv aufzählbar, definiere UTM U, die bei Eingabe $M\#x$, TM M auf x simuliert
2. um zu zeigen, dass MP nicht rekursiv ist, verwende Reduktion N vom Halteproblem nach MP: HP $\leqslant_T$ MP

Satz

MP *ist nicht rekursiv, aber rekursiv aufzählbar*

Beweisskizze.

1. um zu zeigen, dass MP rekursiv aufzählbar, definiere UTM U, die bei Eingabe $M\#x$, TM M auf x simuliert
2. um zu zeigen, dass MP nicht rekursiv ist, verwende Reduktion N vom Halteproblem nach MP: $\mathrm{HP} \leqslant_T \mathrm{MP}$
3. weiters sei K eine totale TM, sodass $\mathrm{MP} = \mathrm{L}(K)$; definiere K' (totale) TM, sodass $\mathrm{HP} = \mathrm{L}(K')$ als Kombination von K und N:

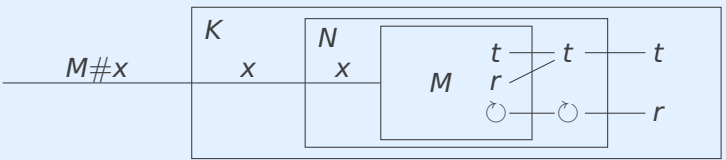


Satz

MP *ist nicht rekursiv, aber rekursiv aufzählbar*

Beweisskizze.

1. um zu zeigen, dass MP rekursiv aufzählbar, definiere UTM U, die bei Eingabe $M\#x$, TM M auf x simuliert
2. um zu zeigen, dass MP nicht rekursiv ist, verwende Reduktion N vom Halteproblem nach MP: $\mathrm{HP} \leqslant_T \mathrm{MP}$
3. weiters sei K eine totale TM, sodass $\mathrm{MP} = \mathrm{L}(K)$; definiere K' (totale) TM, sodass $\mathrm{HP} = \mathrm{L}(K')$ als Kombination von K und N:



Satz

jede rekursive Menge ist rekursiv aufzählbar, aber nicht jede rekursiv aufzählbare Menge ist rekursiv

Satz

jede rekursive Menge ist rekursiv aufzählbar, aber nicht jede rekursiv aufzählbare Menge ist rekursiv ■

Satz

es kann kein Testprogramm für “hello, world” Programme geben

Beweis.

$$\text{HP} \leqslant_T \text{“hello, world” Programme}$$ ■

Satz

jede rekursive Menge ist rekursiv aufzählbar, aber nicht jede rekursiv aufzählbare Menge ist rekursiv ■

Satz

es kann kein Testprogramm für “hello, world” Programme geben

Beweis.

$$\text{HP} \leqslant_T \text{“hello, world” Programme}$$ ■

Satz

die folgenden Probleme sind unentscheidbar:

1. *das Postsche Korrespondenzproblem*
2. *ist eine beliebige Sprache regulär?*

Äquivalente Formulierungen

Definition (informell)

Ein 2-Kellerautomat ist ein endlicher Automat K, der zusätzlich noch zwei Keller (oder Stapel) zur Verfügung hat; $L(K)$ wird mittels einer zweiwertigen Funktion definiert

Äquivalente Formulierungen

Definition (informell)

Ein 2-Kellerautomat ist ein endlicher Automat K, der zusätzlich noch zwei Keller (oder Stapel) zur Verfügung hat; $\mathrm{L}(K)$ wird mittels einer zweiwertigen Funktion definiert

Satz

Sei K ein Kellerautomat, dann existiert eine TM M, sodass $\mathrm{L}(M) = \mathrm{L}(K)$ *und umgekehrt*

Äquivalente Formulierungen

Definition (informell)

Ein 2-Kellerautomat ist ein endlicher Automat K, der zusätzlich noch zwei Keller (oder Stapel) zur Verfügung hat; $\mathrm{L}(K)$ wird mittels einer zweiwertigen Funktion definiert

Satz

Sei K ein Kellerautomat, dann existiert eine TM M, sodass $\mathrm{L}(M) = \mathrm{L}(K)$ *und umgekehrt*

Beweisskizze.

- Simulation eines 2-Kellerautomaten durch eine TM mit drei Bändern: Das obere Band dient der Eingabe und die unteren der Darstellung der Keller
- Andererseits ist TM durch einen 2-Kellerautomaten simulierbar: Der Bandinhalt links vom Schreib-/Lesekopf wird auf dem ersten Keller, der Bandinhalt rechts vom Schreib-/Lesekopf auf dem zweiten Keller dargestellt

Äquivalente Formulierungen

Definition (informell)

Ein 2-Kellerautomat ist ein endlicher Automat K, der zusätzlich noch zwei Keller (oder Stapel) zur Verfügung hat; $\mathrm{L}(K)$ wird mittels einer zweiwertigen Funktion definiert

Satz

Sei K ein Kellerautomat, dann existiert eine TM M, sodass $\mathrm{L}(M) = \mathrm{L}(K)$ *und umgekehrt*

Beweisskizze.

- Simulation eines 2-Kellerautomaten durch eine TM mit drei Bändern: Das obere Band dient der Eingabe und die unteren der Darstellung der Keller
- Andererseits ist TM durch einen 2-Kellerautomaten simulierbar: Der Bandinhalt links vom Schreib-/Lesekopf wird auf dem ersten Keller, der Bandinhalt rechts vom Schreib-/Lesekopf auf dem zweiten Keller dargestellt

Definition (Registermaschine mit goto)

Eine k-Registermaschine mit goto (k-counter machine) ist ein Paar $R = ((x_i)_{1 \leqslant i \leqslant k}, \mathrm{P})$ mit Registern x_i, die natürliche Zahlen beinhalten und ein Programm P; es gibt die folgenden Instruktionen

1. $x_i := x_i + 1$
2. $x_i := x_i - 1$
3. if $x_i = 0$ goto j

Definition (Registermaschine mit goto)

Eine k-Registermaschine mit goto (k-counter machine) ist ein Paar $R = ((x_i)_{1 \leqslant i \leqslant k}, \mathrm{P})$ mit Registern x_i, die natürliche Zahlen beinhalten und ein Programm P; es gibt die folgenden Instruktionen

1. $x_i := x_i + 1$
2. $x_i := x_i - 1$
3. if $x_i = 0$ goto j

Satz

Sei R eine k-Registermaschine (mit goto), dann existiert eine 1-Band TM M, sodass $\mathrm{L}(M) = \mathrm{L}(R)$

Definition (Registermaschine mit goto)

Eine k-Registermaschine mit goto (k-counter machine) ist ein Paar $R = ((x_i)_{1 \leqslant i \leqslant k}, \mathrm{P})$ mit Registern x_i, die natürliche Zahlen beinhalten und ein Programm P; es gibt die folgenden Instruktionen

1. $x_i := x_i + 1$
2. $x_i := x_i - 1$
3. if $x_i = 0$ goto j

Satz

Sei R eine k-Registermaschine (mit goto), dann existiert eine 1-Band TM M, sodass $\mathrm{L}(M) = \mathrm{L}(R)$

Beweisskizze.

R kann am einfachsten durch eine $(k+1)$-bändige Turingmaschine M' simuliert werden: jeder Register auf ein Band ■

Satz

Sei M eine TM, dann existiert eine 4-Registermaschine R, sodass $\mathsf{L}(R) = \mathsf{L}(M)$

Satz

Sei M eine TM, dann existiert eine 4-Registermaschine R, sodass $\mathrm{L}(R) = \mathrm{L}(M)$

Beweis.

- Zunächst simulieren wir M durch eine 2-Kellermaschine K mit Kelleralphabet $\{0, 1\}$
- Danach simulieren jeden Keller durch ein Register, das den dezimalen Wert des Binärstrings am Keller repräsentiert
- Zur Simulation von Stapeloperationen benötigen wir zwei weitere Register
- Hinzufügen von 0 (1) am Keller wird etwa durch Kopie und Verdoppelung (plus 1) erzeugt
- Entfernen von 0 (1) wird durch Rechnen modulo 2 erzeugt

Satz

Sei M eine TM, dann existiert eine 4-Registermaschine R, sodass $\mathrm{L}(R) = \mathrm{L}(M)$

Beweis.

- Zunächst simulieren wir M durch eine 2-Kellermaschine K mit Kelleralphabet $\{0, 1\}$
- Danach simulieren jeden Keller durch ein Register, das den dezimalen Wert des Binärstrings am Keller repräsentiert
- Zur Simulation von Stapeloperationen benötigen wir zwei weitere Register
- Hinzufügen von 0 (1) am Keller wird etwa durch Kopie und Verdoppelung (plus 1) erzeugt
- Entfernen von 0 (1) wird durch Rechnen modulo 2 erzeugt

Definition (Registermaschine mit while, vgl. ETI)

Eine k-Registermaschine mit while R ist ein Paar $R = ((x_i)_{1 \leqslant i \leqslant k}, \mathrm{P})$ mit Registern x_i, die natürliche Zahlen beinhalten und ein Programm P; Instruktionen:

1. $x_i := x_i + 1$
2. $x_i := x_i - 1$
3. while $x_i \neq 0$ do P_1 end

Definition (Registermaschine mit while, vgl. ETI)

Eine k-Registermaschine mit while R ist ein Paar $R = ((x_i)_{1 \leqslant i \leqslant k}, \mathrm{P})$ mit Registern x_i, die natürliche Zahlen beinhalten und ein Programm P; Instruktionen:

1. $x_i := x_i + 1$
2. $x_i := x_i - 1$
3. while $x_i \neq 0$ do P_1 end

Satz

Sei R eine k-Registermaschine mit while, dann existiert eine k–Registermaschine mit goto R', sodass $\mathrm{L}(R) = \mathrm{L}(R')$

Definition (Registermaschine mit while, vgl. ETI)

Eine k-Registermaschine mit while R ist ein Paar $R = ((x_i)_{1 \leqslant i \leqslant k}, \mathrm{P})$ mit Registern x_i, die natürliche Zahlen beinhalten und ein Programm P; Instruktionen:

1. $x_i := x_i + 1$
2. $x_i := x_i - 1$
3. while $x_i \neq 0$ do P_1 end

Satz

Sei R eine k-Registermaschine mit while, dann existiert eine k–Registermaschine mit goto R', sodass $\mathrm{L}(R) = \mathrm{L}(R')$

Beweisskizze.

Jede while Schleife kann durch goto ausgedrückt werden ■

Satz

Sei R eine k-Registermaschine mit goto, dann existiert eine k–Registermaschine R' mit while, sodass $\mathrm{L}(R) = \mathrm{L}(R')$

Satz

Sei R eine k-Registermaschine mit goto, dann existiert eine k–Registermaschine R' mit while, sodass $\mathrm{L}(R) = \mathrm{L}(R')$

Beweis.

Sei P ein Programm einer Registermaschine mit goto; wir schreiben es in ein Programm P' mit while um:

$$
\begin{array}{ll}
1 : B_1; & x_{\text{count}} := 1; \\
2 : B_2; & \text{while } x_{\text{count}} \neq 0 \text{ do} \\
\vdots & \quad \text{if } x_{\text{count}} = 1 \text{ then } B'_1 \text{ end}; \\
n : B_n; & \quad \text{if } x_{\text{count}} = 2 \text{ then } B'_2 \text{ end}; \\
 & \quad \vdots \\
 & \quad \text{if } x_{\text{count}} = n \text{ then } B'_n \text{ end}; \\
 & \quad \text{if } x_{\text{count}} > n \text{ then } x_{\text{count}} := 0 \text{ end};
\end{array}
$$

Äquivalente Formulierungen

Satz

die folgenden Berechnungsmodelle sind äquivalent

- *(deterministische) Turingmaschinen (mit ein- oder zweiseitig unendlichen Bändern)*
- *nichtdeterministische Turingmaschinen*
- *2-Kellerautomaten*
- *Registermaschinen mit* goto
- *Registermaschinen mit* while
- *Aufzählmaschinen*
- *partielle rekursive Funktionen*
- *λ-Kalkül*
- *. . .*

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Resourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Resourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Resourcen

- Speicherplatz
- Rechenzeit
- ...

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Resourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Resourcen

- Speicherplatz
- Rechenzeit
- ...

Problem & Algorithmus

Wir unterscheiden zwischen

- der Komplexität eines Algorithmus
- der Komplexität eines Problems

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Resourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Resourcen

- Speicherplatz
- Rechenzeit
- ...

Problem & Algorithmus

Wir unterscheiden zwischen

- der Komplexität eines Algorithmus — euklidischer Algorithmus: $O(n^2)$ (wobei n die maximale binäre Länge der Eingabe)
- der Komplexität eines Problems

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Resourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Resourcen

- Speicherplatz
- Rechenzeit
- ...

Problem & Algorithmus

Wir unterscheiden zwischen

- der Komplexität eines Algorithmus — euklidischer Algorithmus: $O(n^2)$ (wobei n die maximale binäre Länge der Eingabe)
- der Komplexität eines Problems — Maze ist in P

Definition (Maze)

Gegeben

- gerichteten Graph G mit Eckenmenge E
- Ecken $s, t \in E$

$\exists$ Weg zwischen s und t?

Definition (Maze)

Gegeben

- gerichteten Graph G mit Eckenmenge E
- Ecken $s, t \in E$

$\exists$ Weg zwischen s und t?

Algorithmus

Nachfolgersuche

Definition (Maze)

Gegeben

- gerichteten Graph G mit Eckenmenge E
- Ecken $s, t \in E$

$\exists$ Weg zwischen s und t?

Algorithmus

Nachfolgersuche

Komplexität

- Zeitkomplexität: $O(n^2)$
- Platzbedarf: $O(n)$

wobei $n := \#(E) + \#(K)$

Definition („Hamiltonscher Kreis“ (HK))

∃ Zykel in einem ungerichteten Graph G, der jede Ecke genau einmal enthält (mit Ausnahme der Start-/Endecke)?

Definition („Hamiltonscher Kreis“ (HK))

$\exists$ Zykel in einem ungerichteten Graph G, der jede Ecke genau einmal enthält (mit Ausnahme der Start-/Endecke)?

Algorithmus

Algorithmus könnte für einen ungerichteten Graphen mit Eckenmenge E und Kantenmenge K wie folgt verfahren: Überprüfe für jedes Tupel aus $K^{\#(E)}$, ob es ein Hamiltonscher Kreis ist.

Definition („Hamiltonscher Kreis“ (HK))

$\exists$ Zykel in einem ungerichteten Graph G, der jede Ecke genau einmal enthält (mit Ausnahme der Start-/Endecke)?

Algorithmus

Algorithmus könnte für einen ungerichteten Graphen mit Eckenmenge E und Kantenmenge K wie folgt verfahren: Überprüfe für jedes Tupel aus $K^{\#(E)}$, ob es ein Hamiltonscher Kreis ist.

Komplexität

- Zeitkomplexität: $O(n^{n+2})$
- Platzbedarf: $O(n)$

wobei $n := \#(E) + \#(K)$

Definition („Hamiltonscher Kreis“ (HK))

$\exists$ Zykel in einem ungerichteten Graph G, der jede Ecke genau einmal enthält (mit Ausnahme der Start-/Endecke)?

Algorithmus

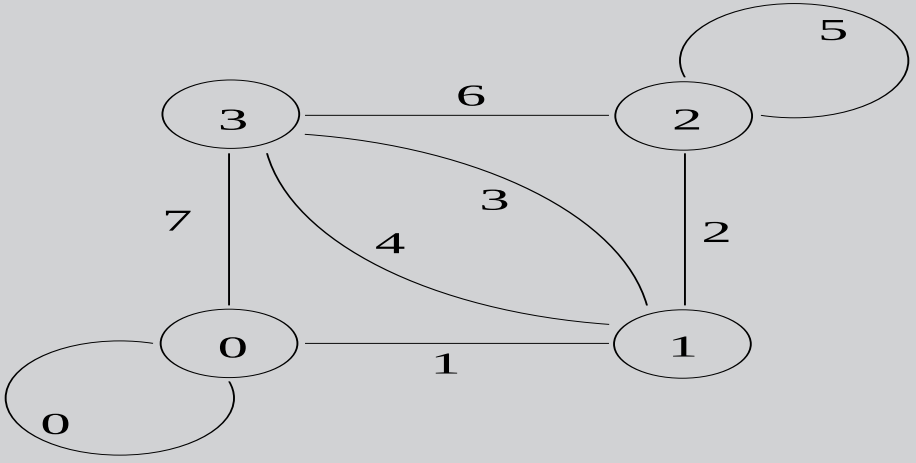
Algorithmus könnte für einen ungerichteten Graphen mit Eckenmenge E und Kantenmenge K wie folgt verfahren: Überprüfe für jedes Tupel aus $K^{\#(E)}$, ob es ein Hamiltonscher Kreis ist.

Komplexität

- Zeitkomplexität: $O(n^{n+2})$
- Platzbedarf: $O(n)$
- Zeitkomplexität kann auf $2^{O(n)}$ verbessert werden

wobei $n := \#(E) + \#(K)$

Beispiel



Dann ist zum Beispiel $(1, 2, 6, 7)$ ein Hamiltonscher Kreis

Definition (“Generalised Geography” (GEO))

das Spiel verallgemeinerte Länderkunde ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

Definition (“Generalised Geography” (GEO))

das Spiel verallgemeinerte Länderkunde ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

1. gerichteter Graph G und Startknoten s

Definition (“Generalised Geography” (GEO))

das Spiel verallgemeinerte Länderkunde ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

1. gerichteter Graph G und Startknoten s
2. Anna beginnt im Land s sie
   wählt ein erreichbares (unmarkiertes) Land t, welches sie markiert

Definition (“Generalised Geography” (GEO))

das Spiel verallgemeinerte Länderkunde ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

1. gerichteter Graph G und Startknoten s
2. Anna beginnt im Land s sie
   wählt ein erreichbares (unmarkiertes) Land t, welches sie markiert
3. Otto zieht von t weiter und markiert wieder

Definition (“Generalised Geography” (GEO))

das Spiel verallgemeinerte Länderkunde ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

1. gerichteter Graph G und Startknoten s
2. Anna beginnt im Land s sie
   wählt ein erreichbares (unmarkiertes) Land t, welches sie markiert
3. Otto zieht von t weiter und markiert wieder
4. derjenige Spieler, der nicht mehr ziehen kann verliert

Definition (“Generalised Geography” (GEO))

das Spiel verallgemeinerte Länderkunde ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

1. gerichteter Graph G und Startknoten s
2. Anna beginnt im Land s sie
   wählt ein erreichbares (unmarkiertes) Land t, welches sie markiert
3. Otto zieht von t weiter und markiert wieder
4. derjenige Spieler, der nicht mehr ziehen kann verliert

$\exists$ Gewinnstrategie für Anna?

Definition (“Generalised Geography” (GEO))

das Spiel verallgemeinerte Länderkunde ist ein Zweipersonenspiel
Spieler heißen Anna und Otto, gegeben

1. gerichteter Graph G und Startknoten s
2. Anna beginnt im Land s sie
   wählt ein erreichbares (unmarkiertes) Land t, welches sie markiert
3. Otto zieht von t weiter und markiert wieder
4. derjenige Spieler, der nicht mehr ziehen kann verliert

$\exists$ Gewinnstrategie für Anna?

Beispiel

Länderkunde

Algorithmus A mit Eingabe (G, b)

1. sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft

Algorithmus A mit Eingabe (G, b)

1. sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
2. eliminiere Knoten b und alle betroffenen Kanten; bezeichne den neuen Graphen mit G'

Algorithmus A mit Eingabe (G, b)

1. sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
2. eliminiere Knoten b und alle betroffenen Kanten; bezeichne den neuen Graphen mit G'
3. $\forall$ unmittelbaren Nachfolger b' von b, rufe A rekursiv mit (G', b') auf

Algorithmus A mit Eingabe (G, b)

1. sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
2. eliminiere Knoten b und alle betroffenen Kanten; bezeichne den neuen Graphen mit G'
3. $\forall$ unmittelbaren Nachfolger b' von b, rufe A rekursiv mit (G', b') auf
4. wenn alle Aufrufe akzeptieren, hat Spieler 2 eine Gewinnstrategie

Algorithmus A mit Eingabe (G, b)

1. sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
2. eliminiere Knoten b und alle betroffenen Kanten; bezeichne den neuen Graphen mit G'
3. $\forall$ unmittelbaren Nachfolger b' von b, rufe A rekursiv mit (G', b') auf
4. wenn alle Aufrufe akzeptieren, hat Spieler 2 eine Gewinnstrategie
5. wenn ein Aufruf nicht akzeptiert, hat Spieler 1 eine Gewinnstrategie; also akzeptiert A

Algorithmus A mit Eingabe (G, b)

1. sei b der aktuelle Knoten und habe dieser Ausgangsgrad 0; Spieler 1 verliert immer und A verwirft
2. eliminiere Knoten b und alle betroffenen Kanten; bezeichne den neuen Graphen mit G'
3. $\forall$ unmittelbaren Nachfolger b' von b, rufe A rekursiv mit (G', b') auf
4. wenn alle Aufrufe akzeptieren, hat Spieler 2 eine Gewinnstrategie
5. wenn ein Aufruf nicht akzeptiert, hat Spieler 1 eine Gewinnstrategie; also akzeptiert A

Komplexität

- Platzkomplexität: $O(n)$
- nur der Rekursionsstack braucht Platz, die Rekursionstiefe ist n

n die Anzahl der Ecken von G

Definition

sei M eine totale DTM oder NTM

- die Laufzeitkomplexität von M ist Funktion $T\colon \mathbb{N} \to \mathbb{N}$, wobei T wie folgt definiert

$$T(n) := \max\{m \mid M \text{ hält bei Eingabe } x,\ \ell(x) = n,\ \text{nach } m \text{ Schritten}\}$$

Definition

sei M eine totale DTM oder NTM

- die Laufzeitkomplexität von M ist Funktion $T\colon \mathbb{N} \to \mathbb{N}$, wobei T wie folgt definiert

$$T(n) := \max\{m \mid M \text{ hält bei Eingabe } x,\ \ell(x) = n,\ \text{nach } m \text{ Schritten}\}$$

- $T(n)$ bezeichnet die Laufzeit von M, wenn n die Länge der Eingabe
- M heißt T-Zeit-Turingmaschine

Definition

sei M eine totale DTM oder NTM

- die Laufzeitkomplexität von M ist Funktion $T\colon \mathbb{N} \to \mathbb{N}$, wobei T wie folgt definiert

$$T(n) := \max\{m \mid M \text{ hält bei Eingabe } x, \ell(x) = n, \text{ nach } m \text{ Schritten}\}$$

- $T(n)$ bezeichnet die Laufzeit von M, wenn n die Länge der Eingabe
- M heißt T-Zeit-Turingmaschine

Definition

sei $T\colon \mathbb{N} \to \mathbb{N}$ eine numerische Funktion

$$\text{DTIME}(T) := \{\mathrm{L}(M) \mid M \text{ ist eine mehrbändige DTM mit Laufzeit in } \mathrm{O}(T)\}$$

$$\text{NTIME}(T) := \{\mathrm{L}(N) \mid N \text{ ist eine mehrbändige NTM mit Laufzeit in } \mathrm{O}(T)\}$$

Beispiel

betrachte Maze als Sprache:

$$\text{Maze} = \{(G, s, t) \mid G \text{ Graph mit Weg von } s \text{ nach } t\}$$

es gilt Maze $\in$ DTIME(n^2)

Beispiel

betrachte Maze als Sprache:

$$\text{Maze} = \{(G, s, t) \mid G \text{ Graph mit Weg von } s \text{ nach } t\}$$

es gilt Maze $\in$ DTIME(n^2)

Beispiel

betrachte HK als Sprache

$$\text{HK} = \{(G, k) \mid \begin{array}{l} G \text{ ist ein ungerichteter Graph mit Hamilton-} \\ \text{schem Kreis } k \text{ }\} \end{array}$$

es gilt HK $\in$ DTIME(n^{n+2}), aber auch HK $\in$ NTIME(n^2)

Beispiel

betrachte Maze als Sprache:

$$\text{Maze} = \{(G, s, t) \mid G \text{ Graph mit Weg von } s \text{ nach } t\}$$

es gilt $\text{Maze} \in \text{DTIME}(n^2)$

Beispiel

betrachte HK als Sprache

$$\text{HK} = \{(G, k) \mid \begin{array}{l} G \text{ ist ein ungerichteter Graph mit Hamilton-} \\ \text{schem Kreis } k \text{ }\} \end{array}$$

es gilt $\text{HK} \in \text{DTIME}(n^{n+2})$, aber auch $\text{HK} \in \text{NTIME}(n^2)$

Definition

$$\text{P} := \bigcup_{k \geqslant 1} \text{DTIME}(n^k) \qquad \text{NP} := \bigcup_{k \geqslant 1} \text{NTIME}(n^k)$$

Definition

- ein Verifikator einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus V sodass

$$L = \{x \in \Sigma^* \mid \exists\, c, \text{ sodass } V \text{ akzeptiert Eingabe } (x, c)\}$$

Definition

- ein Verifikator einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus V sodass
$$L = \{x \in \Sigma^* \mid \exists\, c, \text{ sodass } V \text{ akzeptiert Eingabe } (x, c)\}$$
- ein polytime Verifikator ist ein Verifikator mit Laufzeit $O(n^k)$ wobei $n = \ell(x)$

Definition

- ein Verifikator einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus V sodass
$$L = \{x \in \Sigma^* \mid \exists\, c, \text{ sodass } V \text{ akzeptiert Eingabe } (x, c)\}$$
- ein polytime Verifikator ist ein Verifikator mit Laufzeit $O(n^k)$ wobei $n = \ell(x)$

Definition

NP′ ist die Klasse der Sprachen, die einen polytime Verifikator haben

Definition

- ein Verifikator einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus V sodass
$$L = \{x \in \Sigma^* \mid \exists\, c, \text{ sodass } V \text{ akzeptiert Eingabe } (x, c)\}$$
- ein polytime Verifikator ist ein Verifikator mit Laufzeit $O(n^k)$ wobei $n = \ell(x)$

Definition

NP′ ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte HK; definiere Verifikator V für HK mit Eingabe (G, k)

Definition

- ein Verifikator einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus V sodass
$$L = \{x \in \Sigma^* \mid \exists\, c, \text{ sodass } V \text{ akzeptiert Eingabe } (x, c)\}$$
- ein polytime Verifikator ist ein Verifikator mit Laufzeit $O(n^k)$ wobei $n = \ell(x)$

Definition

NP′ ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte HK; definiere Verifikator V für HK mit Eingabe (G, k)

1. V prüft of k ein Hamiltonscher Kreis ist
2. wenn ja, akzeptiert V, sonst verwirft V
3. V läuft in polynomieller Zeit

Definition

- ein Verifikator einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus V sodass
$$L = \{x \in \Sigma^* \mid \exists\, c, \text{ sodass } V \text{ akzeptiert Eingabe } (x, c)\}$$
- ein polytime Verifikator ist ein Verifikator mit Laufzeit $O(n^k)$ wobei $n = \ell(x)$

Definition

NP′ ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

betrachte HK; definiere Verifikator V für HK mit Eingabe (G, k)

1. V prüft of k ein Hamiltonscher Kreis ist
2. wenn ja, akzeptiert V, sonst verwirft V
3. V läuft in polynomieller Zeit

also gilt HK $\in$ NP′

Lemma

wenn $L \in \mathrm{NP}'$, *dann* $L \in \mathrm{NP}$

Lemma

wenn $L \in \mathrm{NP}'$, *dann* $L \in \mathrm{NP}$

Beweis.

Lemma

wenn $L \in \mathrm{NP}'$, *dann* $L \in \mathrm{NP}$

Beweis.

1. sei V ein Verifikator für L

Lemma

wenn $L \in \mathrm{NP}'$, *dann* $L \in \mathrm{NP}$

Beweis.

1. sei V ein Verifikator für L
2. V läuft in polynomieller Zeit auf Eingabe (x, c),
   das heißt V läuft in Zeit $\mathrm{O}(\ell(x)^k)$ für geeignetes k

Lemma

wenn $L \in \mathrm{NP}'$*, dann* $L \in \mathrm{NP}$

Beweis.

1. sei V ein Verifikator für L
2. V läuft in polynomieller Zeit auf Eingabe (x, c),
   das heißt V läuft in Zeit $\mathrm{O}(\ell(x)^k)$ für geeignetes k
3. definiere NTM N mit Eingabe x
   - rate (nichtdeterministisch) String c mit $\ell(c) = \mathrm{O}(\ell(x)^k)$
   - simuliere V auf (x, c)
   - wenn N akzeptiert, dann akzeptiert V, sonst verwirft V

   Algorithmus ist nichtdeterministisch

Lemma

wenn $L \in \mathrm{NP}'$*, dann* $L \in \mathrm{NP}$

Beweis.

1. sei V ein Verifikator für L
2. V läuft in polynomieller Zeit auf Eingabe (x, c),
   das heißt V läuft in Zeit $\mathrm{O}(\ell(x)^k)$ für geeignetes k
3. definiere NTM N mit Eingabe x
   - rate (nichtdeterministisch) String c mit $\ell(c) = \mathrm{O}(\ell(x)^k)$
   - simuliere V auf (x, c)
   - wenn N akzeptiert, dann akzeptiert V, sonst verwirft V

   Algorithmus ist nichtdeterministisch

Lemma

wenn $L \in \mathrm{NP}'$, *dann* $L \in \mathrm{NP}$

Beweis.

1. sei V ein Verifikator für L
2. V läuft in polynomieller Zeit auf Eingabe (x, c),
   das heißt V läuft in Zeit $\mathrm{O}(\ell(x)^k)$ für geeignetes k
3. definiere NTM N mit Eingabe x
   - rate (nichtdeterministisch) String c mit $\ell(c) = \mathrm{O}(\ell(x)^k)$
   - simuliere V auf (x, c)
   - wenn N akzeptiert, dann akzeptiert V, sonst verwirft V

   Algorithmus ist nichtdeterministisch

■

Beispiel

$\mathrm{HK} \in \mathrm{NP}$

Lemma

wenn $L \in \mathrm{NP}$, *dann* $L \in \mathrm{NP}'$

Lemma

wenn $L \in \mathrm{NP}$, *dann* $L \in \mathrm{NP}'$

Beweis.

Lemma

wenn $L \in \mathrm{NP}$, *dann* $L \in \mathrm{NP}'$

Beweis.

1. sei $L \in \mathrm{NP}$; $\exists$ NTM N, sodass $L = \mathrm{L}(N)$

Lemma

wenn $L \in \mathrm{NP}$, *dann* $L \in \mathrm{NP}'$

Beweis.

1. sei $L \in \mathrm{NP}$; $\exists$ NTM N, sodass $L = \mathrm{L}(N)$
2. N hat polynomielle nichtdeterministische Zeitkomplexität

Lemma

wenn $L \in \mathrm{NP}$, *dann* $L \in \mathrm{NP}'$

Beweis.

1. sei $L \in \mathrm{NP}$; $\exists$ NTM N, sodass $L = \mathrm{L}(N)$
2. N hat polynomielle nichtdeterministische Zeitkomplexität
3. definiere einen polytime Verifikator V mit Eingabe (x, c), wobei c den erfolgreichen Pfad im Berechnungsbaum codiert
   - simuliere N auf x
   - verwende Zertifikat c um Nichtdeterminismus aufzulösen

   Algorithmus ist deterministisch

Lemma

wenn $L \in \mathrm{NP}$, *dann* $L \in \mathrm{NP}'$

Beweis.

1. sei $L \in \mathrm{NP}$; $\exists$ NTM N, sodass $L = \mathrm{L}(N)$
2. N hat polynomielle nichtdeterministische Zeitkomplexität
3. definiere einen polytime Verifikator V mit Eingabe (x, c),
   wobei c den erfolgreichen Pfad im Berechnungsbaum codiert
   - simuliere N auf x
   - verwende Zertifikat c um Nichtdeterminismus aufzulösen

   Algorithmus ist deterministisch
4. wenn N akzeptiert, akzeptiert V, ansonsten verwirft V

Lemma

wenn $L \in \mathrm{NP}$, *dann* $L \in \mathrm{NP}'$

Beweis.

1. sei $L \in \mathrm{NP}$; $\exists$ NTM N, sodass $L = \mathrm{L}(N)$
2. N hat polynomielle nichtdeterministische Zeitkomplexität
3. definiere einen polytime Verifikator V mit Eingabe (x, c),
   wobei c den erfolgreichen Pfad im Berechnungsbaum codiert
   - simuliere N auf x
   - verwende Zertifikat c um Nichtdeterminismus aufzulösen

   Algorithmus ist deterministisch
4. wenn N akzeptiert, akzeptiert V, ansonsten verwirft V

Lemma

wenn $L \in \mathrm{NP}$, *dann* $L \in \mathrm{NP}'$

Beweis.

1. sei $L \in \mathrm{NP}$; $\exists$ NTM N, sodass $L = \mathrm{L}(N)$
2. N hat polynomielle nichtdeterministische Zeitkomplexität
3. definiere einen polytime Verifikator V mit Eingabe (x, c),
   wobei c den erfolgreichen Pfad im Berechnungsbaum codiert
   - simuliere N auf x
   - verwende Zertifikat c um Nichtdeterminismus aufzulösen

   Algorithmus ist deterministisch
4. wenn N akzeptiert, akzeptiert V, ansonsten verwirft V

■

Satz

$\mathrm{NP} = \mathrm{NP}'$

Reduktionen (in polynomieller Zeit)

Definition

1. $\exists$ k-bändige totale DTM M mit Eingabealphabet Σ
2. M läuft in polynomieller Zeit
3. bei Eingabe $x \in \Sigma^*$, schreibt M $R(x)$ auf das (erste) Band

Reduktionen (in polynomieller Zeit)

Definition

1. $\exists$ k-bändige totale DTM M mit Eingabealphabet Σ
2. M läuft in polynomieller Zeit
3. bei Eingabe $x \in \Sigma^*$, schreibt M $R(x)$ auf das (erste) Band

dann heißt $R\colon \Sigma^* \to \Delta^*$ in polynomieller Zeit berechenbar

Reduktionen (in polynomieller Zeit)

Definition

1. $\exists$ k-bändige totale DTM M mit Eingabealphabet Σ
2. M läuft in polynomieller Zeit
3. bei Eingabe $x \in \Sigma^*$, schreibt M $R(x)$ auf das (erste) Band

dann heißt $R\colon \Sigma^* \to \Delta^*$ in polynomieller Zeit berechenbar

Definition

1. $\exists\, R\colon \Sigma^* \to \Delta^*$
2. R berechenbar in polynomieller Zeit
3. für $A \subseteq \Sigma^*$, $B \subseteq \Delta^*$ gilt $x \in A \Leftrightarrow R(x) \in B$

Reduktionen (in polynomieller Zeit)

Definition

1. $\exists$ k-bändige totale DTM M mit Eingabealphabet Σ
2. M läuft in polynomieller Zeit
3. bei Eingabe $x \in \Sigma^*$, schreibt M $R(x)$ auf das (erste) Band

dann heißt $R\colon \Sigma^* \to \Delta^*$ in polynomieller Zeit berechenbar

Definition

1. $\exists\, R\colon \Sigma^* \to \Delta^*$
2. R berechenbar in polynomieller Zeit
3. für $A \subseteq \Sigma^*$, $B \subseteq \Delta^*$ gilt $x \in A \Leftrightarrow R(x) \in B$

dann ist A in polynomieller Zeit auf B reduzierbar; kurz: $A \leqslant^{\mathrm{p}} B$

Beispiel

$$A = \{x \in \{a,b\}^* \mid \ell(x) \text{ ist gerade}\} \text{ und}$$
$$B = \{x \in \{0,1\}^* \mid x \text{ ist ein Palindrom gerader Länge}\}.$$

Dann gilt $A \leqslant^{\mathrm{p}} B$; wir wählen $R(a) = 0$ und $R(b) = 0$; somit wandeln wir ein Wort aus $\{a,b\}^n$ in 0^n um

Beispiel

$$A = \{x \in \{a,b\}^* \mid \ell(x) \text{ ist gerade}\} \text{ und}$$
$$B = \{x \in \{0,1\}^* \mid x \text{ ist ein Palindrom gerader Länge}\}.$$

Dann gilt $A \leqslant^{\mathrm{p}} B$; wir wählen $R(a) = 0$ und $R(b) = 0$; somit wandeln wir ein Wort aus $\{a,b\}^n$ in 0^n um

$x \in A$	x	$R(x)$	$R(x) \in B$
✓	ϵ	ϵ	✓
×	a	0	×
×	b	0	×
✓	aa	00	✓
✓	ab	00	✓
✓	ba	00	✓
⋮	⋮	⋮	⋮

Definition

1. $\mathcal{C}$ eine beliebige Komplexitätsklasse
2. B eine Sprache über Σ und
3. $\forall$ Sprachen $A \in \mathcal{C}$ gilt: $A \leqslant^{\mathrm{p}} B$

Definition

1. $\mathcal{C}$ eine beliebige Komplexitätsklasse
2. B eine Sprache über Σ und
3. $\forall$ Sprachen $A \in \mathcal{C}$ gilt: $A \leqslant^{\mathrm{p}} B$

dann ist $B \leqslant^{\mathrm{p}}$-hart für $\mathcal{C}$

Definition

1. $\mathcal{C}$ eine beliebige Komplexitätsklasse
2. B eine Sprache über Σ und
3. $\forall$ Sprachen $A \in \mathcal{C}$ gilt: $A \leqslant^{\mathrm{p}} B$

dann ist $B \leqslant^{\mathrm{p}}$-hart für $\mathcal{C}$

Beispiel

HK ist $\leqslant^{\mathrm{p}}$-hart für NP

Definition

1. $\mathcal{C}$ eine beliebige Komplexitätsklasse
2. B eine Sprache über Σ und
3. $\forall$ Sprachen $A \in \mathcal{C}$ gilt: $A \leqslant^{\mathrm{p}} B$

dann ist B $\leqslant^{\mathrm{p}}$-hart für $\mathcal{C}$

Beispiel

HK ist $\leqslant^{\mathrm{p}}$-hart für NP

Definition

für eine Sprache B, sei

1. B $\leqslant^{\mathrm{p}}$-hart für $\mathcal{C}$ und
2. $B \in \mathcal{C}$

Definition

1. $\mathcal{C}$ eine beliebige Komplexitätsklasse
2. B eine Sprache über Σ und
3. $\forall$ Sprachen $A \in \mathcal{C}$ gilt: $A \leqslant^{\mathrm{p}} B$

dann ist $B \leqslant^{\mathrm{p}}$-hart für $\mathcal{C}$

Beispiel

HK ist $\leqslant^{\mathrm{p}}$-hart für NP

Definition

für eine Sprache B, sei

1. $B \leqslant^{\mathrm{p}}$-hart für $\mathcal{C}$ und
2. $B \in \mathcal{C}$

dann ist $B \leqslant^{\mathrm{p}}$-vollständig für $\mathcal{C}$ oder (kurz) $\mathcal{C}$-vollständig