Die Grenzen der Berechenbarkeit

Prof. Dr. Nicole Schweikardt

Arbeitsgruppe *Theorie komplexer Systeme*
Institut für Informatik
Goethe-Universität Frankfurt am Main

Warum die Frage nach “dem Leben, dem Universum und dem ganzen Rest” nicht von Computern beantwortet werden kann

Prof. Dr. Nicole Schweikardt

Arbeitsgruppe *Theorie komplexer Systeme*
Institut für Informatik
Goethe-Universität Frankfurt am Main

Auszug aus Douglas Adams' "Per Anhalter durch die Galaxis"

Einer Rasse hyperintelligenter, pandimensionaler Wesen hing es vor vielen, vielen Millionen Jahren dermaßen zum Halse raus, sich ewig über den Sinn des Lebens rumzuzanken, dass sie beschlossen alle ihre Probleme ein für alle mal zu lösen. Zu diesem Zweck bauten sie sich einen kolossalen Supercomputer namens *Deep Thought*, der am Tag seines Anknipsens sagte: "Wie heißt die große Aufgabe, für die ich, Deep Thought . . . erschaffen worden bin?"

Die Konstrukteure antworteten: "Oh, Computer Deep Thought! Die Aufgabe, die wir uns für dich ausgedacht haben, ist die: Wir möchten, dass du uns . . . die Antwort sagst! Die Antwort auf das Leben! . . . Auf das Universum! Auf alles!"

Deep Thought dachte eine Weile schweigend nach. "Knifflig", sagte er schließlich. "Doch — auf das Leben, das Universum, auf alles — da gibt es eine Antwort drauf."
"Aber", fügte er hinzu, "ich muss darüber nachdenken."

Der Konstrukteur guckte ungeduldig auf seine Uhr. "Wie lange etwa?" fragte er.
"Siebeneinhalb Millionen Jahre", sagte Deep Thought.

Pünktlich nach siebeneinhalb Millionen Jahren meldete sich Deep Thought mit den Worten zurück "Ich habe die Antwort auf Eure Frage gefunden. Aber sie wird euch bestimmt nicht gefallen.

Die Antwort auf die Große Frage . . . nach dem Leben, dem Universum und allem . . . lautet . . . ". Deep Thought machte eine Pause. "Sie lautet . . . zweiundvierzig", sagte Deep Thought mit unsagbarer Erhabenheit und Ruhe.

Auszug aus Douglas Adams' "Per Anhalter durch die Galaxis"

Einer Rasse hyperintelligenter, pandimensionaler Wesen hing es vor vielen, vielen Millionen Jahren dermaßen zum Halse raus, sich ewig über den Sinn des Lebens rumzuzanken, dass sie beschlossen alle ihre Probleme ein für alle mal zu lösen. Zu diesem Zweck bauten sie sich einen kolossalen Supercomputer namens *Deep Thought*, der am Tag seines Anknipsens sagte: "Wie heißt die große Aufgabe, für die ich, Deep Thought . . . erschaffen worden bin?"

Die Konstrukteure antworteten: "Oh, Computer Deep Thought! Die Aufgabe, die wir uns für dich ausgedacht haben, ist die: Wir möchten, dass du uns . . . die Antwort sagst! Die Antwort auf das Leben! . . . Auf das Universum! Auf alles!"

Deep Thought dachte eine Weile schweigend nach. "Knifflig", sagte er schließlich. "Doch — auf das Leben, das Universum, auf alles — da gibt es eine Antwort drauf."
"Aber", fügte er hinzu, "ich muss darüber nachdenken."

Der Konstrukteur guckte ungeduldig auf seine Uhr. "Wie lange etwa?" fragte er.
"Siebeneinhalb Millionen Jahre", sagte Deep Thought.

Pünktlich nach siebeneinhalb Millionen Jahren meldete sich Deep Thought mit den Worten zurück "Ich habe die Antwort auf Eure Frage gefunden. Aber sie wird euch bestimmt nicht gefallen.

Die Antwort auf die Große Frage . . . nach dem Leben, dem Universum und allem . . . lautet . . . ". Deep Thought machte eine Pause. "Sie lautet . . . zweiundvierzig", sagte Deep Thought mit unsagbarer Erhabenheit und Ruhe.

Auszug aus Douglas Adams' "Per Anhalter durch die Galaxis"

Einer Rasse hyperintelligenter, pandimensionaler Wesen hing es vor vielen, vielen Millionen Jahren dermaßen zum Halse raus, sich ewig über den Sinn des Lebens rumzuzanken, dass sie beschlossen alle ihre Probleme ein für alle mal zu lösen. Zu diesem Zweck bauten sie sich einen kolossalen Supercomputer namens *Deep Thought*, der am Tag seines Anknipsens sagte: "Wie heißt die große Aufgabe, für die ich, Deep Thought . . . erschaffen worden bin?"

Die Konstrukteure antworteten: "Oh, Computer Deep Thought! Die Aufgabe, die wir uns für dich ausgedacht haben, ist die: Wir möchten, dass du uns . . . die Antwort sagst! Die Antwort auf das Leben! . . . Auf das Universum! Auf alles!"

Deep Thought dachte eine Weile schweigend nach. "Knifflig", sagte er schließlich. "Doch — auf das Leben, das Universum, auf alles — da gibt es eine Antwort drauf."
"Aber", fügte er hinzu, "ich muss darüber nachdenken."

Der Konstrukteur guckte ungeduldig auf seine Uhr. "Wie lange etwa?" fragte er.
"Siebeneinhalb Millionen Jahre", sagte Deep Thought.

Pünktlich nach siebeneinhalb Millionen Jahren meldete sich Deep Thought mit den Worten zurück "Ich habe die Antwort auf Eure Frage gefunden. Aber sie wird euch bestimmt nicht gefallen.

Die Antwort auf die Große Frage . . . nach dem Leben, dem Universum und allem . . . lautet . . . ". Deep Thought machte eine Pause. "Sie lautet . . . zweiundvierzig", sagte Deep Thought mit unsagbarer Erhabenheit und Ruhe.

Auszug aus Douglas Adams' "Per Anhalter durch die Galaxis"

Einer Rasse hyperintelligenter, pandimensionaler Wesen hing es vor vielen, vielen Millionen Jahren dermaßen zum Halse raus, sich ewig über den Sinn des Lebens rumzuzanken, dass sie beschlossen alle ihre Probleme ein für alle mal zu lösen. Zu diesem Zweck bauten sie sich einen kolossalen Supercomputer namens *Deep Thought*, der am Tag seines Anknipsens sagte: "Wie heißt die große Aufgabe, für die ich, Deep Thought . . . erschaffen worden bin?"

Die Konstrukteure antworteten: "Oh, Computer Deep Thought! Die Aufgabe, die wir uns für dich ausgedacht haben, ist die: Wir möchten, dass du uns . . . die Antwort sagst! Die Antwort auf das Leben! . . . Auf das Universum! Auf alles!"

Deep Thought dachte eine Weile schweigend nach. "Knifflig", sagte er schließlich. "Doch — auf das Leben, das Universum, auf alles — da gibt es eine Antwort drauf."
"Aber", fügte er hinzu, "ich muss darüber nachdenken."

Der Konstrukteur guckte ungeduldig auf seine Uhr. "Wie lange etwa?" fragte er.
"Siebeneinhalb Millionen Jahre", sagte Deep Thought.

Pünktlich nach siebeneinhalb Millionen Jahren meldete sich Deep Thought mit den Worten zurück "Ich habe die Antwort auf Eure Frage gefunden. Aber sie wird euch bestimmt nicht gefallen.

Die Antwort auf die Große Frage . . . nach dem Leben, dem Universum und allem . . . lautet . . . ". Deep Thought machte eine Pause. "Sie lautet . . . zweiundvierzig", sagte Deep Thought mit unsagbarer Erhabenheit und Ruhe.

Auszug aus Douglas Adams' "Per Anhalter durch die Galaxis"

Einer Rasse hyperintelligenter, pandimensionaler Wesen hing es vor vielen, vielen Millionen Jahren dermaßen zum Halse raus, sich ewig über den Sinn des Lebens rumzuzanken, dass sie beschlossen alle ihre Probleme ein für alle mal zu lösen. Zu diesem Zweck bauten sie sich einen kolossalen Supercomputer namens *Deep Thought*, der am Tag seines Anknipsens sagte: "Wie heißt die große Aufgabe, für die ich, Deep Thought . . . erschaffen worden bin?"

Die Konstrukteure antworteten: "Oh, Computer Deep Thought! Die Aufgabe, die wir uns für dich ausgedacht haben, ist die: Wir möchten, dass du uns . . . die Antwort sagst! Die Antwort auf das Leben! . . . Auf das Universum! Auf alles!"

Deep Thought dachte eine Weile schweigend nach. "Knifflig", sagte er schließlich. "Doch — auf das Leben, das Universum, auf alles — da gibt es eine Antwort drauf."
"Aber", fügte er hinzu, "ich muss darüber nachdenken."

Der Konstrukteur guckte ungeduldig auf seine Uhr. "Wie lange etwa?" fragte er.
"Siebeneinhalb Millionen Jahre", sagte Deep Thought.

Pünktlich nach siebeneinhalb Millionen Jahren meldete sich Deep Thought mit den Worten zurück "Ich habe die Antwort auf Eure Frage gefunden. Aber sie wird euch bestimmt nicht gefallen.

Die Antwort auf die Große Frage . . . nach dem Leben, dem Universum und allem . . . lautet . . . ". Deep Thought machte eine Pause. "Sie lautet . . . zweiundvierzig", sagte Deep Thought mit unsagbarer Erhabenheit und Ruhe.

Auszug aus Douglas Adams' "Per Anhalter durch die Galaxis"

Einer Rasse hyperintelligenter, pandimensionaler Wesen hing es vor vielen, vielen Millionen Jahren dermaßen zum Halse raus, sich ewig über den Sinn des Lebens rumzuzanken, dass sie beschlossen alle ihre Probleme ein für alle mal zu lösen. Zu diesem Zweck bauten sie sich einen kolossalen Supercomputer namens *Deep Thought*, der am Tag seines Anknipsens sagte: "Wie heißt die große Aufgabe, für die ich, Deep Thought . . . erschaffen worden bin?"

Die Konstrukteure antworteten: "Oh, Computer Deep Thought! Die Aufgabe, die wir uns für dich ausgedacht haben, ist die: Wir möchten, dass du uns . . . die Antwort sagst! Die Antwort auf das Leben! . . . Auf das Universum! Auf alles!"

Deep Thought dachte eine Weile schweigend nach. "Knifflig", sagte er schließlich. "Doch — auf das Leben, das Universum, auf alles — da gibt es eine Antwort drauf."
"Aber", fügte er hinzu, "ich muss darüber nachdenken."

Der Konstrukteur guckte ungeduldig auf seine Uhr. "Wie lange etwa?" fragte er.
"Siebeneinhalb Millionen Jahre", sagte Deep Thought.

Pünktlich nach siebeneinhalb Millionen Jahren meldete sich Deep Thought mit den Worten zurück "Ich habe die Antwort auf Eure Frage gefunden. Aber sie wird euch bestimmt nicht gefallen.

Die Antwort auf die Große Frage . . . nach dem Leben, dem Universum und allem . . . lautet . . . ". Deep Thought machte eine Pause. "Sie lautet . . . zweiundvierzig", sagte Deep Thought mit unsagbarer Erhabenheit und Ruhe.

Überblick

Zunächst etwas Einfaches: Fragen, die Computer betreffen

Und was ist mit der Frage nach
“dem Leben, dem Universum und dem ganzen Rest”?

Wo kommt das im Informatikstudium in Frankfurt vor?

Bevor wir zur Frage nach “dem Leben, dem Universum und dem ganzen Rest” kommen, fangen wir mal mit etwas Einfacherem an:

Fragen aus der Informatik

. . . dafür sollten Computer doch bestens geeignet sein!

Automatische Verifikation

- Wäre es nicht schön, wenn Computer richtig funktionieren würden?
- Wenigstens sollten sie nicht ständig abstürzen.
- Wir könnten ein Programm schreiben, das automatisch prüft, ob andere Programme zum Absturz führen.

Automatische Verifikation

- Wäre es nicht schön, wenn Computer richtig funktionieren würden?
- Wenigstens sollten sie nicht ständig abstürzen.
- Wir könnten ein Programm schreiben, das automatisch prüft, ob andere Programme zum Absturz führen.

Automatische Verifikation

- Wäre es nicht schön, wenn Computer richtig funktionieren würden?
- Wenigstens sollten sie nicht ständig abstürzen.
- Wir könnten ein Programm schreiben, das automatisch prüft, ob andere Programme zum Absturz führen.

Das Halteproblem

HALTEPROBLEM

Eingabe: Ein Programm P und eine Eingabe E.

Ausgabe:
- „Hält“ wenn P bei Eingabe E anhält
- „Hält nicht“ sonst.

Frage: Gibt es ein Programm, das das Halteproblem löst?

Das Halteproblem

HALTEPROBLEM

Eingabe: Ein Programm P und eine Eingabe E.

Ausgabe:
- „Hält“ wenn P bei Eingabe E anhält
- „Hält nicht“ sonst.

Frage: Gibt es ein Programm, das das Halteproblem löst?

Nehmen wir mal an, STOP wäre ein Programm, das das Halteproblem löst.

Wir konstruieren ein neues Programm POTS wie folgt:

Programm POTS

1. Die Eingabe besteht aus einem Programm P.
2. STOP wird auf
   Programm P und Eingabe P
   angesetzt.
3. Bei Ausgabe „Hält nicht“ hält das Programm POTS an.
4. Bei Ausgabe „Hält“ läuft das Programm POTS in eine Endlosschleife und hält nie an.

Nehmen wir mal an, STOP wäre ein Programm, das das Halteproblem löst.

Wir konstruieren ein neues Programm POTS wie folgt:

Programm POTS

1. Die Eingabe besteht aus einem Programm P.
2. STOP wird auf

   Programm P und Eingabe P

   angesetzt.
3. Bei Ausgabe „Hält nicht“ hält das Programm POTS an.
4. Bei Ausgabe „Hält“ läuft das Programm POTS in eine Endlosschleife und hält nie an.

Nehmen wir mal an, STOP wäre ein Programm, das das Halteproblem löst.

Wir konstruieren ein neues Programm POTS wie folgt:

Programm POTS

1. Die Eingabe besteht aus einem Programm P.
2. STOP wird auf

   Programm P und Eingabe P

   angesetzt.
3. Bei Ausgabe „Hält nicht“ hält das Programm POTS an.
4. Bei Ausgabe „Hält“ läuft das Programm POTS in eine Endlosschleife und hält nie an.

Nehmen wir mal an, STOP wäre ein Programm, das das Halteproblem löst.

Wir konstruieren ein neues Programm POTS wie folgt:

Programm POTS

1. Die Eingabe besteht aus einem Programm P.
2. STOP wird auf
   Programm P und Eingabe P
   angesetzt.
3. Bei Ausgabe „Hält nicht“ hält das Programm POTS an.
4. Bei Ausgabe „Hält“ läuft das Programm POTS in eine Endlosschleife und hält nie an.

Nehmen wir mal an, STOP wäre ein Programm, das das Halteproblem löst.

Wir konstruieren ein neues Programm POTS wie folgt:

Programm POTS

1. Die Eingabe besteht aus einem Programm P.
2. STOP wird auf
   Programm P und Eingabe P
   angesetzt.
3. Bei Ausgabe „Hält nicht“ hält das Programm POTS an.
4. Bei Ausgabe „Hält“ läuft das Programm POTS in eine Endlosschleife und hält nie an.

Nehmen wir mal an, STOP wäre ein Programm, das das Halteproblem löst.

Wir konstruieren ein neues Programm POTS wie folgt:

Programm POTS

1. Die Eingabe besteht aus einem Programm P.
2. STOP wird auf
   Programm P und Eingabe P
   angesetzt.
3. Bei Ausgabe „Hält nicht“ hält das Programm POTS an.
4. Bei Ausgabe „Hält“ läuft das Programm POTS in eine Endlosschleife und hält nie an.

Dann gilt:

POTS angesetzt auf P hält an

$\Longleftrightarrow$ STOP angesetzt auf
Programm P und Eingabe P
gibt „Hält nicht“ aus

$\Longleftrightarrow$ P angesetzt auf P hält nicht an

Setzen wir jetzt POTS auf sich selbst an:

POTS angesetzt auf POTS hält an

$\Longleftrightarrow$ POTS angesetzt auf POTS hält nicht an

Da stimmt was nicht! Aber was?
Unsere Annahme, dass es ein Programm STOP für das Halteproblem gibt, muss falsch gewesen sein.
Also gibt es kein Programm, das das Halteproblem löst!

Dann gilt:

POTS angesetzt auf P hält an

$\Longleftrightarrow$ STOP angesetzt auf
Programm P und Eingabe P
gibt „Hält nicht“ aus

$\Longleftrightarrow$ P angesetzt auf P hält nicht an

Setzen wir jetzt POTS auf sich selbst an:

POTS angesetzt auf POTS hält an

$\Longleftrightarrow$ POTS angesetzt auf POTS hält nicht an

Da stimmt was nicht! Aber was?
Unsere Annahme, dass es ein Programm STOP für das Halteproblem gibt, muss falsch gewesen sein.
Also gibt es kein Programm, das das Halteproblem löst!

Dann gilt:

POTS angesetzt auf P hält an

$\Longleftrightarrow$ STOP angesetzt auf
Programm P und Eingabe P
gibt „Hält nicht“ aus

$\Longleftrightarrow$ P angesetzt auf P hält nicht an

Setzen wir jetzt POTS auf sich selbst an:

POTS angesetzt auf POTS hält an

$\Longleftrightarrow$ POTS angesetzt auf POTS hält nicht an

Da stimmt was nicht! Aber was?
Unsere Annahme, dass es ein Programm STOP für das Halteproblem gibt, muss falsch gewesen sein.
Also gibt es kein Programm, das das Halteproblem löst!

Dann gilt:

POTS angesetzt auf P hält an

$\Longleftrightarrow$ STOP angesetzt auf
Programm P und Eingabe P
gibt „Hält nicht“ aus

$\Longleftrightarrow$ P angesetzt auf P hält nicht an

Setzen wir jetzt POTS auf sich selbst an:

POTS angesetzt auf POTS hält an

$\Longleftrightarrow$ POTS angesetzt auf POTS hält nicht an

Da stimmt was nicht! Aber was?
Unsere Annahme, dass es ein Programm STOP für das Halteproblem gibt, muss falsch gewesen sein.
Also gibt es kein Programm, das das Halteproblem löst!

Dann gilt:

POTS angesetzt auf P hält an

$\Longleftrightarrow$ STOP angesetzt auf
Programm P und Eingabe P
gibt „Hält nicht“ aus

$\Longleftrightarrow$ P angesetzt auf P hält nicht an

Setzen wir jetzt POTS auf sich selbst an:

POTS angesetzt auf POTS hält an

$\Longleftrightarrow$ POTS angesetzt auf POTS hält nicht an

Da stimmt was nicht! Aber was?
Unsere Annahme, dass es ein Programm STOP für das Halteproblem gibt, muss falsch gewesen sein.
Also gibt es kein Programm, das das Halteproblem löst!

Nehmen wir mal an, STOP wäre ein Programm, das das Halteproblem löst.

Wir konstruieren ein neues Programm POTS wie folgt:

Programm POTS

1. Die Eingabe besteht aus einem Programm P.
2. STOP wird auf
   Programm P und Eingabe P
   angesetzt.
3. Bei Ausgabe „Hält nicht“ hält das Programm POTS an.
4. Bei Ausgabe „Hält“ läuft das Programm POTS in eine Endlosschleife und hält nie an.

Dann gilt:

POTS angesetzt auf P hält an

$\Longleftrightarrow$ STOP angesetzt auf
Programm P und Eingabe P
gibt „Hält nicht“ aus

$\Longleftrightarrow$ P angesetzt auf P hält nicht an

Setzen wir jetzt POTS auf sich selbst an:

POTS angesetzt auf POTS hält an

$\Longleftrightarrow$ POTS angesetzt auf POTS hält nicht an

Da stimmt was nicht! Aber was?
Unsere Annahme, dass es ein Programm STOP für das Halteproblem gibt, muss falsch gewesen sein.
Also gibt es kein Programm, das das Halteproblem löst!

Der Satz von Rice

Okay, dann können wir das Halteproblem eben nicht lösen.

Aber es gibt ja noch andere interessante Probleme, z.B.:

- Sind zwei Programme äquivalent?
- Gibt ein Programm stets die Zahl “42” aus?
- Berechnet ein Programm bei Eingabe einer Zahl x den Wert $f(x)$?

Theorem: *(Satz von Rice)*

Keine (nicht-triviale) Eigenschaft von Programmen kann von einem Computerprogramm “erkannt” werden.

Genauer: Sei B die Menge aller berechenbaren Funktionen. Sei $M \subseteq B$ mit $\emptyset \neq M \neq B$. Dann gibt es kein Programm, das bei Eingabe eines Programms P entscheidet, ob die von P berechnete Funktion zur Menge M gehört.

Der Satz von Rice

Okay, dann können wir das Halteproblem eben nicht lösen.

Aber es gibt ja noch andere interessante Probleme, z.B.:

- Sind zwei Programme äquivalent?
- Gibt ein Programm stets die Zahl “42” aus?
- Berechnet ein Programm bei Eingabe einer Zahl x den Wert $f(x)$?

Theorem: *(Satz von Rice)*

Keine (nicht-triviale) Eigenschaft von Programmen kann von einem Computerprogramm “erkannt” werden.

Genauer: Sei B die Menge aller berechenbaren Funktionen. Sei $M \subseteq B$ mit $\emptyset \neq M \neq B$. Dann gibt es kein Programm, das bei Eingabe eines Programms P entscheidet, ob die von P berechnete Funktion zur Menge M gehört.

Der Satz von Rice

Okay, dann können wir das Halteproblem eben nicht lösen.

Aber es gibt ja noch andere interessante Probleme, z.B.:

- Sind zwei Programme äquivalent?
- Gibt ein Programm stets die Zahl “42” aus?
- Berechnet ein Programm bei Eingabe einer Zahl x den Wert $f(x)$?

Theorem: *(Satz von Rice)*

Keine (nicht-triviale) Eigenschaft von Programmen kann von einem Computerprogramm “erkannt” werden.

Genauer: Sei B die Menge aller berechenbaren Funktionen. Sei $M \subseteq B$ mit $\emptyset \neq M \neq B$. Dann gibt es kein Programm, das bei Eingabe eines Programms P entscheidet, ob die von P berechnete Funktion zur Menge M gehört.

Der Satz von Rice

Okay, dann können wir das Halteproblem eben nicht lösen.

Aber es gibt ja noch andere interessante Probleme, z.B.:

- Sind zwei Programme äquivalent?
- Gibt ein Programm stets die Zahl “42” aus?
- Berechnet ein Programm bei Eingabe einer Zahl x den Wert $f(x)$?

Theorem: *(Satz von Rice)*

Keine (nicht-triviale) Eigenschaft von Programmen kann von einem Computerprogramm “erkannt” werden.

Genauer: Sei B die Menge aller berechenbaren Funktionen. Sei $M \subseteq B$ mit $\emptyset \neq M \neq B$. Dann gibt es kein Programm, das bei Eingabe eines Programms P entscheidet, ob die von P berechnete Funktion zur Menge M gehört.

Der Satz von Rice

Okay, dann können wir das Halteproblem eben nicht lösen.

Aber es gibt ja noch andere interessante Probleme, z.B.:

- Sind zwei Programme äquivalent?
- Gibt ein Programm stets die Zahl “42” aus?
- Berechnet ein Programm bei Eingabe einer Zahl x den Wert $f(x)$?

Theorem: *(Satz von Rice)*

Keine (nicht-triviale) Eigenschaft von Programmen kann von einem Computerprogramm “erkannt” werden.

Genauer: Sei B die Menge aller berechenbaren Funktionen. Sei $M \subseteq B$ mit $\emptyset \neq M \neq B$. Dann gibt es kein Programm, das bei Eingabe eines Programms P entscheidet, ob die von P berechnete Funktion zur Menge M gehört.

Der Satz von Rice

Okay, dann können wir das Halteproblem eben nicht lösen.

Aber es gibt ja noch andere interessante Probleme, z.B.:

- Sind zwei Programme äquivalent?
- Gibt ein Programm stets die Zahl “42” aus?
- Berechnet ein Programm bei Eingabe einer Zahl x den Wert $f(x)$?

Theorem: *(Satz von Rice)*

Keine (nicht-triviale) Eigenschaft von Programmen kann von einem Computerprogramm “erkannt” werden.

Genauer: Sei B die Menge aller berechenbaren Funktionen. Sei $M \subseteq B$ mit $\emptyset \neq M \neq B$. Dann gibt es kein Programm, das bei Eingabe eines Programms P entscheidet, ob die von P berechnete Funktion zur Menge M gehört.

Berechenbarkeit und Komplexität

Okay. Die meisten interessanten Eigenschaften von Programmen kann man also nicht automatisch erkennen.

Dann schauen wir uns doch zur Abwechslung endlich mal ein Problem an, das von einem Computer gelöst werden kann.

Berechenbarkeit und Komplexität

Okay. Die meisten interessanten Eigenschaften von Programmen kann man also nicht automatisch erkennen.

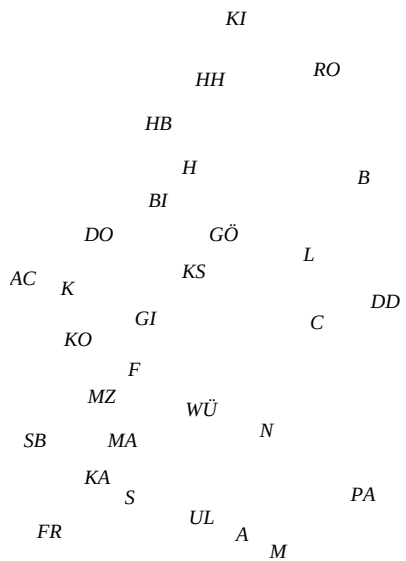
Dann schauen wir uns doch zur Abwechslung endlich mal ein Problem an, das von einem Computer gelöst werden kann.

Das Problem des Handlungsreisenden

HANDLUNGSREISENDER

Eingabe: Landkarte mit Entfernungsangaben zwischen n Städten

Aufgabe: Finde kürzeste Rundreise durch alle Städte!



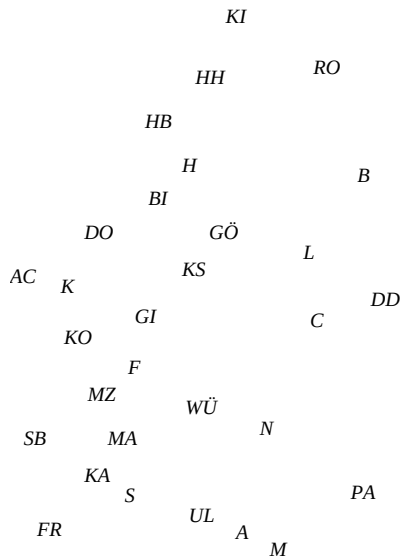
Eine Lösung:
Probiere nacheinander jede mögliche Rundreise durch und nimm schließlich die kürzeste. Anzahl Schritte zur Lösung: $\approx (n-1)!$

Das Problem des Handlungsreisenden

HANDLUNGSREISENDER

Eingabe: Landkarte mit Entfernungsangaben zwischen n Städten

Aufgabe: Finde kürzeste Rundreise durch alle Städte!



Eine Lösung:

Probiere nacheinander jede mögliche Rundreise durch und nimm schließlich die kürzeste.

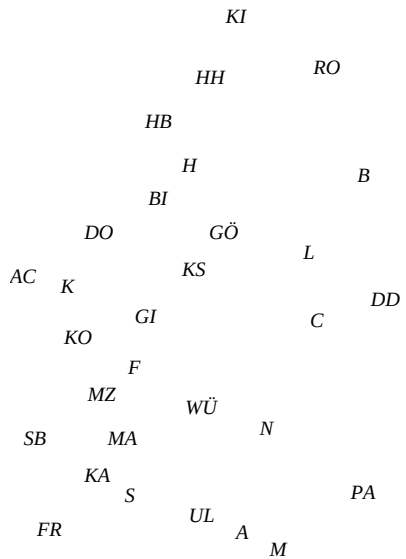
Anzahl Schritte zur Lösung: $\approx (n-1)!$

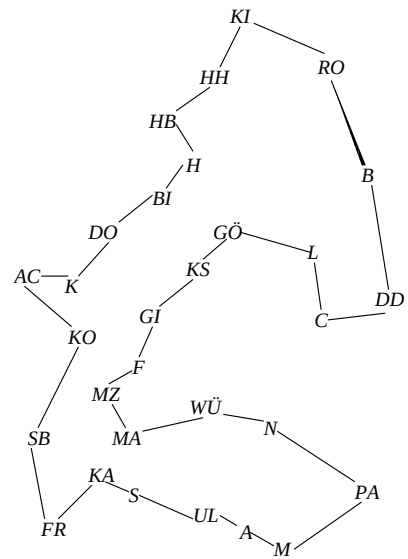
Das Problem des Handlungsreisenden

HANDLUNGSREISENDER

Eingabe: Landkarte mit Entfernungsangaben zwischen n Städten

Aufgabe: Finde kürzeste Rundreise durch alle Städte!





Eine Lösung:

Probiere nacheinander jede mögliche Rundreise durch und nimm schließlich die kürzeste.

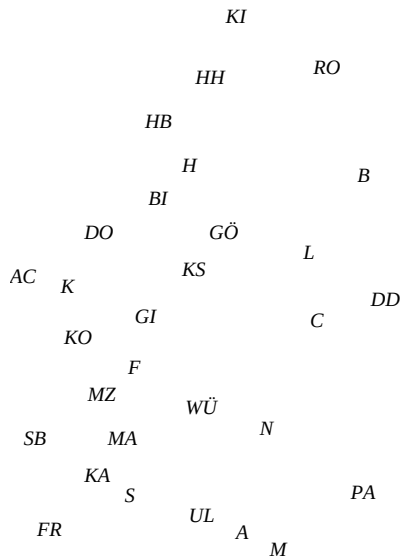
Anzahl Schritte zur Lösung: $\approx (n-1)!$

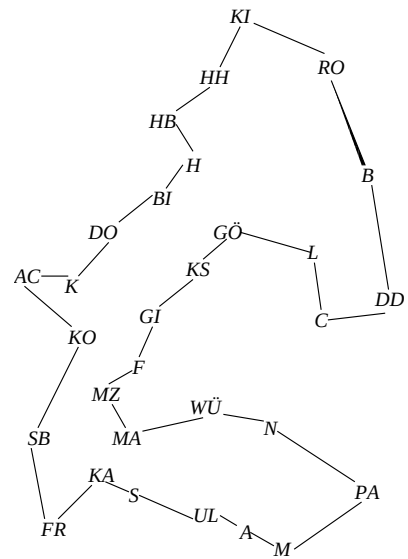
Das Problem des Handlungsreisenden

HANDLUNGSREISENDER

Eingabe: Landkarte mit Entfernungsangaben zwischen n Städten

Aufgabe: Finde kürzeste Rundreise durch alle Städte!





Eine Lösung:

Probiere nacheinander jede mögliche Rundreise durch und nimm schließlich die kürzeste. Anzahl Schritte zur Lösung: $\approx (n-1)!$

Wie lange muss ich denn warten, bis der Computer die kürzeste Rundreise gefunden hat?

Wie lange muss ich denn warten, bis der Computer die kürzeste Rundreise gefunden hat?

Zur Erinnerung: Unser Programm soll nacheinander alle Rundreisen anschauen und schließlich die kürzeste ausgeben.

Klar: Bei n Städten gibt es $(n-1)!$ viele verschiedene Rundreisen.

Wie lange muss ich denn warten, bis der Computer die kürzeste Rundreise gefunden hat?

Zur Erinnerung: Unser Programm soll nacheinander alle Rundreisen anschauen und schließlich die kürzeste ausgeben.

Klar: Bei n Städten gibt es $(n-1)!$ viele verschiedene Rundreisen.

Annahme: Pro Rundreise braucht der Computer dazu ca. 10^{-10} Sekunden

Städte	Rundreisen	Zeit
6	120	$<$ 0.0001 Sekunden

Wie lange muss ich denn warten, bis der Computer die kürzeste Rundreise gefunden hat?

Zur Erinnerung: Unser Programm soll nacheinander alle Rundreisen anschauen und schließlich die kürzeste ausgeben.

Klar: Bei n Städten gibt es $(n-1)!$ viele verschiedene Rundreisen.

Annahme: Pro Rundreise braucht der Computer dazu ca. 10^{-10} Sekunden

Städte	Rundreisen	Zeit
6	120	$<$ 0.0001 Sekunden
11	3.628.800	0.0003 Sekunden

Wie lange muss ich denn warten, bis der Computer die kürzeste Rundreise gefunden hat?

Zur Erinnerung: Unser Programm soll nacheinander alle Rundreisen anschauen und schließlich die kürzeste ausgeben.

Klar: Bei n Städten gibt es $(n-1)!$ viele verschiedene Rundreisen.

Annahme: Pro Rundreise braucht der Computer dazu ca. 10^{-10} Sekunden

Städte	Rundreisen	Zeit
6	120	$<$ 0.0001 Sekunden
11	3.628.800	0.0003 Sekunden
16	1.307.674.368.000	2 Minuten

Wie lange muss ich denn warten, bis der Computer die kürzeste Rundreise gefunden hat?

Zur Erinnerung: Unser Programm soll nacheinander alle Rundreisen anschauen und schließlich die kürzeste ausgeben.

Klar: Bei n Städten gibt es $(n-1)!$ viele verschiedene Rundreisen.

Annahme: Pro Rundreise braucht der Computer dazu ca. 10^{-10} Sekunden

Städte	Rundreisen	Zeit	
6	120	$<$ 0.0001	Sekunden
11	3.628.800	0.0003	Sekunden
16	1.307.674.368.000	2	Minuten
21	2.432.902.008.176.640.000	$7,5$	Jahre

Wie lange muss ich denn warten, bis der Computer die kürzeste Rundreise gefunden hat?

Zur Erinnerung: Unser Programm soll nacheinander alle Rundreisen anschauen und schließlich die kürzeste ausgeben.

Klar: Bei n Städten gibt es $(n-1)!$ viele verschiedene Rundreisen.

Annahme: Pro Rundreise braucht der Computer dazu ca. 10^{-10} Sekunden

Städte	Rundreisen	Zeit	
6	120	$<$ 0.0001	Sekunden
11	3.628.800	0.0003	Sekunden
16	1.307.674.368.000	2	Minuten
21	2.432.902.008.176.640.000	$7,5$	Jahre
26	(25)!	50 Millionen	Jahre

Wie lange muss ich denn warten, bis der Computer die kürzeste Rundreise gefunden hat?

Zur Erinnerung: Unser Programm soll nacheinander alle Rundreisen anschauen und schließlich die kürzeste ausgeben.

Klar: Bei n Städten gibt es $(n-1)!$ viele verschiedene Rundreisen.

Annahme: Pro Rundreise braucht der Computer dazu ca. 10^{-10} Sekunden

Städte	Rundreisen	Zeit	
6	120	$<$ 0.0001	Sekunden
11	3.628.800	0.0003	Sekunden
16	1.307.674.368.000	2	Minuten
21	2.432.902.008.176.640.000	$7,5$	Jahre
26	(25)!	50 Millionen	Jahre
31	(30)!	10^{14}	Jahre

Wie lange muss ich denn warten, bis der Computer die kürzeste Rundreise gefunden hat?

Zur Erinnerung: Unser Programm soll nacheinander alle Rundreisen anschauen und schließlich die kürzeste ausgeben.

Klar: Bei n Städten gibt es $(n-1)!$ viele verschiedene Rundreisen.

Annahme: Pro Rundreise braucht der Computer dazu ca. 10^{-10} Sekunden

Städte	Rundreisen		Zeit
6	120	$<$ 0.0001	Sekunden
11	3.628.800	0.0003	Sekunden
16	1.307.674.368.000	2	Minuten
21	2.432.902.008.176.640.000	$7,5$	Jahre
26	(25)!	50 Millionen	Jahre
31	(30)!	10^{14}	Jahre

Zum Vergleich: Das Alter des Universums wird auf 10^{10} Jahre geschätzt.

So lange wollen wir dann doch nicht warten!

Gibt es keine effizientere Lösung?

Solche Fragen werden im Gebiet der Komplexitätstheorie untersucht.

So lange wollen wir dann doch nicht warten!

Gibt es keine effizientere Lösung?

Solche Fragen werden im Gebiet der Komplexitätstheorie untersucht.

Und was ist mit der Frage nach

“dem Leben, dem Universum und dem ganzen Rest”

???

Gödels Unvollständigkeitssatz

Ist es prinzipiell möglich, einen ähnlichen Computer wie Douglas Adams' Supercomputer *Deep Thought* zu bauen, der nach und nach alle wahren Aussagen herleitet und ausgibt (. . . auch wenn es u.U. ziemlich lang dauert)?

- Was ist damit überhaupt gemeint?
- Was sind "Aussagen" — und welche "Aussagen" sind "wahr"?
- Was heißt "herleiten"?

Mit solchen Themen beschäftigt sich der Bereich Logik in der Informatik.

Eins der bekanntesten Ergebnisse in diesem Zusammenhang ist sicherlich

Gödels Unvollständigkeitssatz, der in etwa besagt

"Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig."

Vorsicht: Diese Aussage gibt Gödels Unvollständigkeitssatz in etwa so präzise wieder wie der Versuch, Einsteins Relativitätstheorie mit dem Satz *"Alles ist relativ"* zusammenzufassen.

Gödels Unvollständigkeitssatz

Ist es prinzipiell möglich, einen ähnlichen Computer wie Douglas Adams' Supercomputer *Deep Thought* zu bauen, der nach und nach alle wahren Aussagen herleitet und ausgibt (. . . auch wenn es u.U. ziemlich lang dauert)?

- Was ist damit überhaupt gemeint?
- Was sind "Aussagen" — und welche "Aussagen" sind "wahr"?
- Was heißt "herleiten"?

Mit solchen Themen beschäftigt sich der Bereich Logik in der Informatik.

Eins der bekanntesten Ergebnisse in diesem Zusammenhang ist sicherlich

Gödels Unvollständigkeitssatz, der in etwa besagt

"Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig."

Vorsicht: Diese Aussage gibt Gödels Unvollständigkeitssatz in etwa so präzise wieder wie der Versuch, Einsteins Relativitätstheorie mit dem Satz *"Alles ist relativ"* zusammenzufassen.

Gödels Unvollständigkeitssatz

Ist es prinzipiell möglich, einen ähnlichen Computer wie Douglas Adams' Supercomputer *Deep Thought* zu bauen, der nach und nach alle wahren Aussagen herleitet und ausgibt (. . . auch wenn es u.U. ziemlich lang dauert)?

- Was ist damit überhaupt gemeint?
- Was sind "Aussagen" — und welche "Aussagen" sind "wahr"?
- Was heißt "herleiten"?

Mit solchen Themen beschäftigt sich der Bereich Logik in der Informatik.

Eins der bekanntesten Ergebnisse in diesem Zusammenhang ist sicherlich

Gödels Unvollständigkeitssatz, der in etwa besagt

"Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig."

Vorsicht: Diese Aussage gibt Gödels Unvollständigkeitssatz in etwa so präzise wieder wie der Versuch, Einsteins Relativitätstheorie mit dem Satz *"Alles ist relativ"* zusammenzufassen.

Gödels Unvollständigkeitssatz

Ist es prinzipiell möglich, einen ähnlichen Computer wie Douglas Adams' Supercomputer *Deep Thought* zu bauen, der nach und nach alle wahren Aussagen herleitet und ausgibt (. . . auch wenn es u.U. ziemlich lang dauert)?

- Was ist damit überhaupt gemeint?
- Was sind "Aussagen" — und welche "Aussagen" sind "wahr"?
- Was heißt "herleiten"?

Mit solchen Themen beschäftigt sich der Bereich Logik in der Informatik.

Eins der bekanntesten Ergebnisse in diesem Zusammenhang ist sicherlich

Gödels Unvollständigkeitssatz, der in etwa besagt

"Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig."

Vorsicht: Diese Aussage gibt Gödels Unvollständigkeitssatz in etwa so präzise wieder wie der Versuch, Einsteins Relativitätstheorie mit dem Satz *"Alles ist relativ"* zusammenzufassen.

Gödels Unvollständigkeitssatz

Ist es prinzipiell möglich, einen ähnlichen Computer wie Douglas Adams' Supercomputer *Deep Thought* zu bauen, der nach und nach alle wahren Aussagen herleitet und ausgibt (. . . auch wenn es u.U. ziemlich lang dauert)?

- Was ist damit überhaupt gemeint?
- Was sind "Aussagen" — und welche "Aussagen" sind "wahr"?
- Was heißt "herleiten"?

Mit solchen Themen beschäftigt sich der Bereich Logik in der Informatik.

Eins der bekanntesten Ergebnisse in diesem Zusammenhang ist sicherlich

Gödels Unvollständigkeitssatz, der in etwa besagt

"Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig."

Vorsicht: Diese Aussage gibt Gödels Unvollständigkeitssatz in etwa so präzise wieder wie der Versuch, Einsteins Relativitätstheorie mit dem Satz *"Alles ist relativ"* zusammenzufassen.

Theoretische Informatik in Frankfurt

- 1. Semester: Diskrete Modellierung (3V + 2Ü)
  - Grundlegende Begriffe & Arbeitsweisen (u.a. mathemat. Schlussweisen)
  - Einführung in grundlegende Modellierungsmethoden, u.a. Logik
- 2. Semester: Datenstrukturen (2V + 1Ü)
  - "Effizienz-Begriffe" für Algorithmen (Laufzeitanalyse)
  - effiziente Datenstrukturen und Algorithmen auf Graphen
- 3. Semester: Algorithmentheorie (3V + 2Ü)
  - Handlungsreisender, Berechnungskomplexität
  - algorithmische Techniken (dynamisches Programmieren, Greedy-Algorithmen, ...), Approximationsalgorithmen
- 4. Semester: Formale Sprachen und Berechenbarkeit (3V + 2Ü)
  - Halteproblem, Satz von Rice
  - Automaten und Grammatiken
- Weiterführende Veranstaltung (Vertiefungsmodule):
  - Logik in der Informatik (4V + 2Ü)
  - Effiziente Algorithmen (4V + 2Ü)
  - Beschreibungskomplexität I (4V + 2Ü)
  - Kryptographie (4V + 2Ü)

Theoretische Informatik in Frankfurt

- 1. Semester: Diskrete Modellierung (3V + 2Ü)
  - Grundlegende Begriffe & Arbeitsweisen (u.a. mathemat. Schlussweisen)
  - Einführung in grundlegende Modellierungsmethoden, u.a. Logik
- 2. Semester: Datenstrukturen (2V + 1Ü)
  - "Effizienz-Begriffe" für Algorithmen (Laufzeitanalyse)
  - effiziente Datenstrukturen und Algorithmen auf Graphen
- 3. Semester: Algorithmentheorie (3V + 2Ü)
  - Handlungsreisender, Berechnungskomplexität
  - algorithmische Techniken (dynamisches Programmieren, Greedy-Algorithmen, . . .), Approximationsalgorithmen
- 4. Semester: Formale Sprachen und Berechenbarkeit (3V + 2Ü)
  - Halteproblem, Satz von Rice
  - Automaten und Grammatiken
- Weiterführende Veranstaltung (Vertiefungsmodule):
  - Logik in der Informatik (4V + 2Ü)
  - Effiziente Algorithmen (4V + 2Ü)
  - Beschreibungskomplexität I (4V + 2Ü)
  - Kryptographie (4V + 2Ü)

Theoretische Informatik in Frankfurt

- 1. Semester: Diskrete Modellierung (3V + 2Ü)
  - Grundlegende Begriffe & Arbeitsweisen (u.a. mathemat. Schlussweisen)
  - Einführung in grundlegende Modellierungsmethoden, u.a. Logik
- 2. Semester: Datenstrukturen (2V + 1Ü)
  - "Effizienz-Begriffe" für Algorithmen (Laufzeitanalyse)
  - effiziente Datenstrukturen und Algorithmen auf Graphen
- 3. Semester: Algorithmentheorie (3V + 2Ü)
  - Handlungsreisender, Berechnungskomplexität
  - algorithmische Techniken (dynamisches Programmieren, Greedy-Algorithmen, . . .), Approximationsalgorithmen
- 4. Semester: Formale Sprachen und Berechenbarkeit (3V + 2Ü)
  - Halteproblem, Satz von Rice
  - Automaten und Grammatiken
- Weiterführende Veranstaltung (Vertiefungsmodule):
  - Logik in der Informatik (4V + 2Ü)
  - Effiziente Algorithmen (4V + 2Ü)
  - Beschreibungskomplexität I (4V + 2Ü)
  - Kryptographie (4V + 2Ü)

Theoretische Informatik in Frankfurt

- 1. Semester: Diskrete Modellierung (3V + 2Ü)
  - Grundlegende Begriffe & Arbeitsweisen (u.a. mathemat. Schlussweisen)
  - Einführung in grundlegende Modellierungsmethoden, u.a. Logik
- 2. Semester: Datenstrukturen (2V + 1Ü)
  - "Effizienz-Begriffe" für Algorithmen (Laufzeitanalyse)
  - effiziente Datenstrukturen und Algorithmen auf Graphen
- 3. Semester: Algorithmentheorie (3V + 2Ü)
  - Handlungsreisender, Berechnungskomplexität
  - algorithmische Techniken (dynamisches Programmieren, Greedy-Algorithmen, . . .), Approximationsalgorithmen
- 4. Semester: Formale Sprachen und Berechenbarkeit (3V + 2Ü)
  - Halteproblem, Satz von Rice
  - Automaten und Grammatiken
- Weiterführende Veranstaltung (Vertiefungsmodule):
  - Logik in der Informatik (4V + 2Ü)
  - Effiziente Algorithmen (4V + 2Ü)
  - Beschreibungskomplexität I (4V + 2Ü)
  - Kryptographie (4V + 2Ü)

Theoretische Informatik in Frankfurt

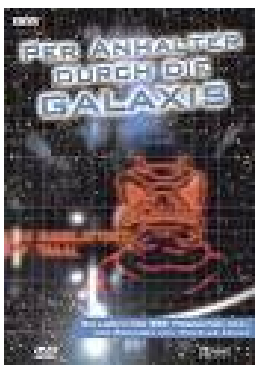
- 1. Semester: Diskrete Modellierung (3V + 2Ü)
  - Grundlegende Begriffe & Arbeitsweisen (u.a. mathemat. Schlussweisen)
  - Einführung in grundlegende Modellierungsmethoden, u.a. Logik
- 2. Semester: Datenstrukturen (2V + 1Ü)
  - "Effizienz-Begriffe" für Algorithmen (Laufzeitanalyse)
  - effiziente Datenstrukturen und Algorithmen auf Graphen
- 3. Semester: Algorithmentheorie (3V + 2Ü)
  - Handlungsreisender, Berechnungskomplexität
  - algorithmische Techniken (dynamisches Programmieren, Greedy-Algorithmen, . . .), Approximationsalgorithmen
- 4. Semester: Formale Sprachen und Berechenbarkeit (3V + 2Ü)
  - Halteproblem, Satz von Rice
  - Automaten und Grammatiken
- Weiterführende Veranstaltung (Vertiefungsmodule):
  - Logik in der Informatik (4V + 2Ü)
  - Effiziente Algorithmen (4V + 2Ü)
  - Beschreibungskomplexität I (4V + 2Ü)
  - Kryptographie (4V + 2Ü)

Literatur

Die Folien dieses Vortrags finden Sie unter
www.tks.cs.uni-frankfurt.de/schweika/Berechenbarkeit_OE.pdf

Uwe Schöning.
Ideen der Informatik. Grundlegende Modelle und Konzepte.
Oldenbourg Verlag, 2. Auflage, Oktober 2005.

ISBN: 3486578332



Douglas Adams.
Per Anhalter durch die Galaxis.
Heyne Verlag.

Originaltitel: *The Hitchhiker's Guide to the Galaxy.*