Grundbegriffe der Theoretischen Informatik

Sommersemester 2017 - Beate Bollig

Die Folien basieren auf den Materialien von Thomas Schwentick.

Teil C: Berechenbarkeit und Entscheidbarkeit

14: Unentscheidbare Probleme 1

Einleitung

- Wir beschäftigen uns in diesem Kapitel mit
  - algorithmischen Problemen, die nicht entscheidbar sind, und
  - dem Beweis dieser Tatsache
- Dabei lernen wir zwei Beweismethoden kennen:
  - Diagonalisierung
  - Reduktion
- Wir illustrieren das Prinzip zuerst an einem informellen Beispiel, bevor wir uns den „richtigen“ Sätzen und Beweisen zuwenden

Inhalt

„hello, world“-Programme: Einleitung (1/3)

Beispiel: „hello, world“-Programm in Java

```
class HelloWorld {
    static public void main( String args[ ] ) {
        System.out.println( „Hello World!“ );
    }
}
```

- **„hello, world“-Programme** werden oft als erstes Beispiel beim Lehren einer Programmiersprache verwendet

- „hello, world“-Programme in Hunderten von Programmiersprachen finden sich auf **helloworldcollection.github.io**

Beispiel: „hello, world“-Programm in C++

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[]) {
cout « „Hello World!“ « endl;
return 0;
}
```

Beispiel: „hello, world“-Programm in Oz

```
functor
import
    System
    Application
define
    {System.showInfo „Hello World!“}
    {Application.exit 0}
end
```

„hello, world“-Programme: Einleitung (2/3)

- Für unsere Zwecke sind die syntaktischen Details konkreter Programmiersprachen nicht so wichtig
- Wir beschreiben Programme deshalb in Pseudocode

Beispiel: „hello, world“-Programm in Pseudocode

```
BEGIN
    PRINT(„hello, world“)
END
```

Definition

- Ein **„hello, world“-Programm (hw-Programm)** sei ein Programm, das keine Eingabe erwartet und als erstes „hello, world“ ausgibt

- **Frage:** Wie schwierig ist es, einem Programm anzusehen, ob es ein „hello, world“-Programm ist?
- **Was könnte daran schwierig sein???**

„hello, world“-Programme: Einleitung (3/3)

Beispiel: „hello, world“-Programm?

$m := 3$
while TRUE **do**
  for $n := 3$ TO m **do**
    for $x := 1$ TO m **do**
      for $y := 1$ TO m **do**
        for z := 1 TO m **do**
          if $x^n + y^n = z^n$ **then**
            PRINT(„hello, world“)
  $m := m + 1$

- Dieses Programm sucht systematisch natürliche Zahlen n, x, y, z mit
  - $n \geqslant 3$ und $x^n + y^n = z^n$
- Wenn es solche Zahlen gibt, wird irgendwann „hello, world“ ausgegeben

✎ Zur Erinnerung: Natürliche Zahlen in dieser Vorlesung: $1, 2, 3, \ldots$

Satz von Fermat

- Es gibt keine natürlichen Zahlen $x, y, z \in \mathbb{N}$ und $n \geqslant 3$ mit $$x^n + y^n = z^n$$

✎ Der Beweis dieses Satzes hat 350 Jahre gedauert...

Korollar

- Das Beispielprogramm ist kein „hello, world“-Programm

- Warum ist es so schwierig herauszufinden, ob dieses Programm ein „hello, world“-Programm ist?

- **Intuitive Schwierigkeit:** Im Beispiel-Programm gibt es unendlich viele Wertekombinationen für x, y, z, n
- Diese können nicht in endlicher Zeit ausprobiert werden

„hello, world“-Tester: Definition

- Herauszufinden, ob ein gegebenes Programm ein „hello, world“-Programm ist, ist also nicht ganz so leicht
- Aber wir haben ja Computer!

- Programme sind Zeichenketten (Strings) und können von anderen Programmen als Eingabe eingelesen werden
- Schreiben wir also einfach ein Programm, das automatisch testet, ob ein gegebenes Programm ein „hello, world“-Programm ist

Definition: „hello, world“-Problem

Gegeben: Programm P

Frage: Ist P ein „hello, world“-Programm?

- Wir nennen ein Programm für das „hello, world“-Problem einen „hello, world“-Tester
  - Ein „hello, world“-Tester gibt also bei Eingabe eines Programmes P die Antwort
    - „ja“, falls P **ein** „hello, world“-Programm ist
    - „nein“, falls P **kein** „hello, world“-Programm ist

- Ein „hello, world“-Tester würde also herausfinden, dass das zweite Beispiel-Programm kein „hello, world“-Programm ist

➨ „hello, world“-Tester müssen ziemlich clever programmiert sein

- **Gibt es überhaupt „hello, world“-Tester?**
- Falls es keine „hello, world“-Tester gibt, **lässt sich das beweisen?**

„hello, world“-Tester: Theorem (1/5)

Theorem

- Es gibt keine „hello, world“-Tester

- Wir beweisen zuerst, dass es keine Tester für das folgende (scheinbar etwas schwierigere) Problem für Programme *mit Eingaben* gibt
- Danach zeigen wir, dass es dann auch keine „hello, world“-Tester gibt

Definition: hw-Problem mit Eingabe

Gegeben: Programm P, Eingabe I

Frage: Gibt P bei Eingabe I „hello, world“ aus?

- Vereinbarung:
  - Programme lesen ihre Eingabe mit Anweisungen der Art „s := READ“
  - Jede solche Anweisung liest den jeweils nächsten String der Eingabe

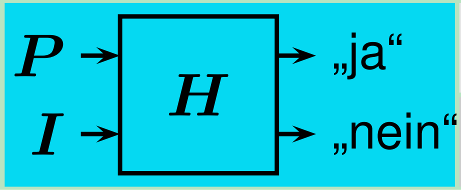
- Wir beweisen jetzt zuerst, dass es keinen Tester für das „hello, world“-Problem mit Eingabe gibt

- Wir führen einen Bewei durch Widerspruch:
  - Wir nehmen an, es gäbe einen solchen Tester
  - Wir zeigen, dass sich daraus ein Widerspruch ergibt
  - Wir schließen daraus, dass die Annahme, es gäbe einen solchen Tester, falsch ist

„hello, world“-Tester: Theorem (2/5)

„Beweis“

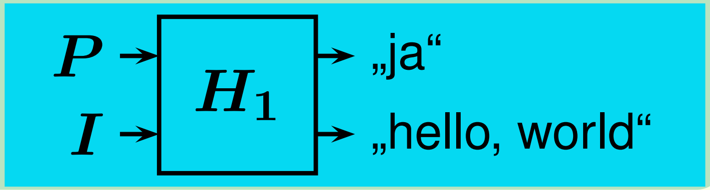
- Annahme: es gibt einen Tester H für das „hello, world“-Problem mit Eingabe:



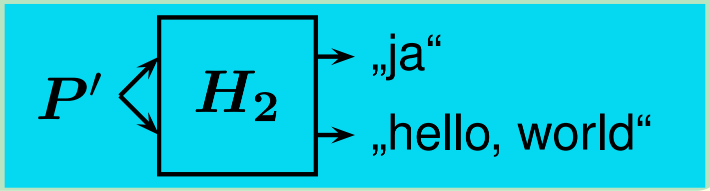
falls P bei Eingabe I „hello, world“ ausgibt

andernfalls

- Wir können H in ein Programm H_1 ändern, das wie H arbeitet, aber „hello, world“ anstelle von „nein“ ausgibt:

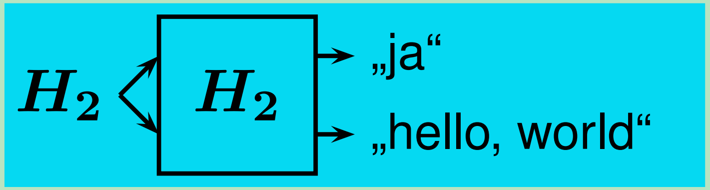


- Wir können H_1 in ein Programm H_2 ändern, das sich bei Eingabe eines Programmes P' so verhält wie H_1 bei Eingabe P' (für P) und P' (für I):



„Beweis“ (Forts.)

- **Wie verhält sich H_2 bei Eingabe H_2?**



✎ Notation: $H(P, I) \stackrel{\text{def}}{=}$ Ausgabe von H bei Eingabe P und I

- **1. Fall:** $H_2(H_2) =$ „ja“
  - ➡ $H_1(H_2, H_2) =$ „ja“
  - ➡ $H(H_2, H_2) =$ „ja“

➡ $H(H_2, H_2)$ ist falsch, denn:
  - ∗ H_2 gibt bei Eingabe H_2 nicht „hello, world“ aus

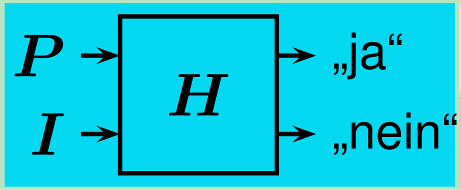
➡ **Widerspruch** zur Annahme, dass H ein Tester für das „hello, world“-Problem mit Eingabe ist

- Der erste Fall kann also nicht eintreten

„hello, world“-Tester: Theorem (3/5)

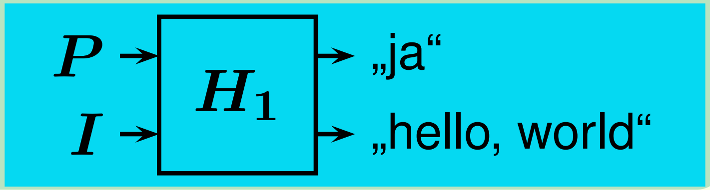
„Beweis“ (Forts.)

- Annahme: es gibt einen Tester H für das „hello, world“-Problem mit Eingabe:

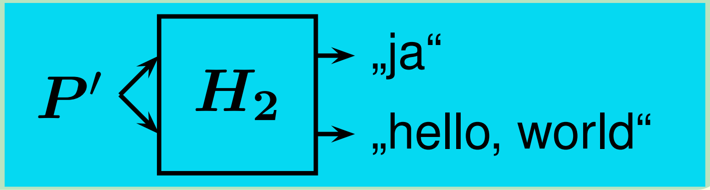


falls P bei Eingabe I „hello, world“ ausgibt
andernfalls

- Wir können H in ein Programm H_1 ändern, das wie H arbeitet, aber „hello, world“ anstelle von „nein“ ausgibt:

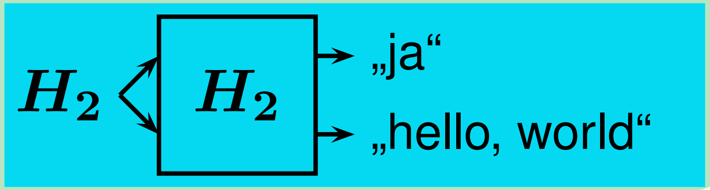


- Wir können H_1 in ein Programm H_2 ändern, das sich bei Eingabe eines Programmes P' so verhält wie H_1 bei Eingabe P' (für P) und P' (für I):



„Beweis“ (Forts.)

- **Wie verhält sich H_2 bei Eingabe H_2?**



- **2. Fall:** $H_2(H_2) =$ „hello, world“
  - ➡ $H_1(H_2, H_2) =$ „hello, world“
  - ➡ $H(H_2, H_2) =$ „nein“

- ➡ $H(H_2, H_2)$ ist falsch, denn:
  - $*$ H_2 gibt bei Eingabe H_2 „hello, world“ aus
- ➡ **Widerspruch** zur Annahme, dass H ein Tester H für das „hello, world“-Problem mit Eingabe ist
- Der zweite Fall kann also auch nicht eintreten

- Die Annahme der Existenz eines Testers H führt also zu einem **Widerspruch**
- ➡ Ein solcher Tester existiert nicht

„hello, world“-Tester: Theorem (4/5)

„Beweis“ (Forts.)

- Es gibt also keine Tester für hw-Programme mit Eingabe
- Dass es auch keine „hello, world“-Tester (für Programme ohne Eingabe) gibt, beweisen wir durch eine *Reduktion*

- Wir zeigen:
  - Wenn es einen „hello, world“-Tester H' (für Programme ohne Eingabe) gäbe, dann auch einen Tester H für das „hello, world“-Problem mit Eingabe

„Beweis“ (Forts.)

- Denn um zu testen, ob ein Programm P mit Eingabe I „hello, world“ ausgibt, könnte H wie folgt vorgehen

- Konstruiere aus P ein Programm P_I ohne Eingabe:
  - Ersetze dazu die Anweisung „$s :=$ READ“ durch „$s := I$“
- Teste mit Hilfe von H', ob P_I „hello, world“ ausgibt
- Falls „ja“: Ausgabe „ja“
- Falls „nein“: Ausgabe „nein“

- Da wir aber schon bewiesen haben, dass es keinen Tester für das „hello, world“-Problem mit Eingabe gibt, gibt es auch keinen Tester für das „hello, world“-Problem

„hello, world“-Tester: „Theorem“ (5/5)

- Die Begriffe „Theorem“ und „beweisen“ stehen auf den vorhergehenden Folien in Anführungszeichen:
  - Um aus den Überlegungen der beiden letzten Folien wirklich ein Theorem und einen Beweis zu erhalten, müssten die verwendeten Begriffe präzise mathematische Definitionen haben

- Die Beweisidee lässt sich jedoch auf unsere formal definierten Berechnungsmodelle übertragen
- Denn der Beweis verwendet im Wesentlichen, dass Programme sich auf einfache Weise modifizieren lassen, z.B.:
  - Modifikation der Ausgabe
  - Initialisierung des Programms mit einer Eingabe (statt Lesen der Eingabe)

- Wir werden nun zeigen, dass ein konkretes algorithmisches Problem, das auf Turingmaschinen basiert, unentscheidbar ist, und danach mit Hilfe von Reduktionen die Unentscheidbarkeit (vieler) anderer Probleme nachweisen

Inhalt

Die „Diagonalsprache“ TM-DIAG (1/2)

- Wir beweisen jetzt für ein erstes konkretes Problem, dass es unentscheidbar ist

- Der Beweis verläuft ähnlich wie der informelle Beweis, dass es kein Programm zur Lösung des „hello, world“-Problems gibt
- Statt für Programme mit Eingabe zu fragen, ob sie „hello, world“ ausgeben, werden wir für Turingmaschinen M fragen, ob sie ihre eigene Kodierung durch einen String akzeptieren

- Im Folgenden betrachten wir Turingmaschinen ausschließlich über dem Ein-/Ausgabealphabet $\boldsymbol{\Sigma = \{0, 1\}}$
  - Die Resultate gelten aber entsprechend auch für jedes andere feste Alphabet

Die „Diagonalsprache“ TM-DIAG (2/2)

- Wir nehmen im Folgenden an, dass wir eine Kodierung von Turingmaschinen zur Verfügung haben, die die folgenden Eigenschaften hat:
  - Für jede TM M gibt es einen String enc(M), der sie kodiert
  - Jeder String w kodiert eine TM M_w

    ✎ Syntaktisch sinnlose Strings kodieren die TM $M_\_$, die immer sofort anhält und ablehnt
- Wie eine solche Kodierung konkret aussehen kann, betrachten wir in Kapitel 16

Definition: TM-DIAG

Gegeben: Turingmaschine M

Frage: Akzeptiert M die Eingabe enc(M)?

Satz 14.1

- TM-DIAG ist nicht entscheidbar

- Der Beweis verwendet die Methode der **Diagonalisierung**

- TM-DIAG scheint kein besonders interessantes algorithmisches Problem zu sein
  - Warum sollte es uns interessieren, ob eine TM „sich selbst“ akzeptiert?

- Das Resultat, dass TM-DIAG unentscheidbar ist, ist nur Mittel zum Zweck:
  - Wir werden alle weiteren Unentscheidbarkeitsresultate auf die Unentscheidbarkeit von TM-DIAG zurückführen

TM-DIAG **ist unentscheidbar (1/3)**

- Im Beweis, dass TM-DIAG unentscheidbar ist, verwenden wir die folgende Aufzählung aller Strings über Σ^*
  - $v_1 = \epsilon, v_2 = 0, v_3 = 1, v_4 = 00, \ldots$
- Statt M_{v_i} schreiben wir M_i
  - $M_1, M_2, M_3, \ldots$ ist also eine Aufzählung aller Turingmaschinen und für jedes i mit $M_i \neq M_-$ gilt: $\text{enc}(M_i) = v_i$

➨ $L_{\text{TM-DIAG}} = \{v_i \mid M_i \text{ akzeptiert die Eingabe } v_i\}$

TM-DIAG **ist unentscheidbar (2/3)**

Illustration der Beweisidee

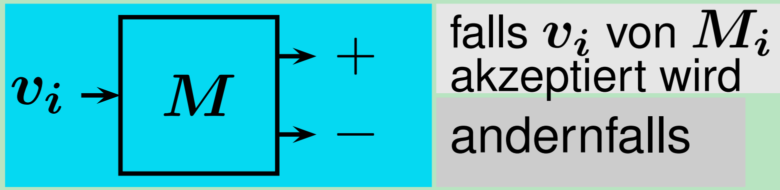
- Wir betrachten das Akzeptier- und Terminationsverhalten von M_i bei Eingabe v_j für alle Kombinationen von i und j:

	v_1	v_2	v_3	v_4	v_5	$\cdots$
M'	$-$	$+$	$-$	$+$	$-$	$\cdots$
M	$+$	$-$	$+$	$-$	$+$	$\cdots$
M_1	$+$	$-$	$\perp$	$+$	$-$	$\cdots$
M_2	$+$	$-$	$+$	$\perp$	$-$	$\cdots$
M_3	$\perp$	$-$	$+$	$-$	$+$	$\cdots$
M_4	$-$	$+$	$+$	$\perp$	$-$	$\cdots$
M_5	$+$	$-$	$+$	$-$	$+$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

- $+$: M_i akzeptiert v_j
- $-$: M_i lehnt v_j ab
- $\perp$: M_i läuft bei Eingabe v_j endlos

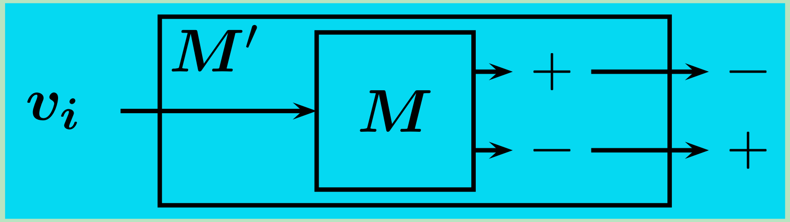
Illustration der Beweisidee (Forts.)

- Annahme: es gibt eine TM M für TM-DIAG:



  - M hält immer an und akzeptiert v_i genau dann, wenn v_i von M_i akzeptiert wird

- Wir modifizieren M zu M' durch Umkehr des Akzeptierverhaltens:



- Dann gibt es ℓ mit $M_\ell = M'$ $(\neq M_{-})$

➨ Dann sind äquivalent:
  - M_ℓ akzeptiert v_ℓ
  - M' akzeptiert v_ℓ ☞ $M' = M_\ell$
  - M akzeptiert v_ℓ nicht

➨ Widerspruch

➨ TM-DIAG ist nicht entscheidbar!

TM-DIAG ist unentscheidbar (3/3)

- Wir beschreiben den Beweis nun noch einmal etwas ausführlicher

Beweisskizze zu „TM-DIAG nicht entscheidbar"

- Um einen Widerspruch zu erreichen, nehmen wir an, M wäre ein Turingmaschine, die TM-DIAG entscheidet
  - Zur Erinnerung: M müsste für alle Eingaben w anhalten und die richtige Antwort geben

- Sei M' die Turingmaschine, die bei Eingabe w zuerst M bei Eingabe w simuliert und dann
  - akzeptiert, falls M ablehnt, und
  - ablehnt, falls M akzeptiert
- Da M für jede Eingabe anhält (und akzeptiert oder ablehnt), gilt dies auch für M'

Beweisskizze (Forts.)

- 1. Fall: $M' \in$ TM-DIAG
  - ➡ M akzeptiert enc(M')
    ☞ nach Annahme über M
  - ➡ M' lehnt enc(M') ab
    ☞ nach Konstruktion von M'
  - ➡ $M' \notin$ TM-DIAG
    ☞ nach Definition von TM-DIAG
  - ➡ Widerspruch

- 2. Fall: $M' \notin$ TM-DIAG
  - ➡ M akzeptiert enc(M') nicht
    ☞ nach Annahme über M
  - ➡ M' akzeptiert enc(M')
    ☞ nach Konstruktion von M'
  - ➡ $M' \in$ TM-DIAG
    ☞ nach Definition von TM-DIAG
  - ➡ Widerspruch

- In beiden Fällen ergibt sich ein Widerspruch

➡ TM-DIAG ist nicht entscheidbar

Bedeutung des Begriffs Unentscheidbarkeit

- Wichtiger Hinweis:
  - Dass TM-DIAG unentscheidbar ist, bedeutet nur, dass es kein *allgemeines Verfahren* gibt, das für *alle* Eingaben M terminiert und entscheidet, ob M die Eingabe enc(M) akzeptiert
  - Für viele Turingmaschinen M lässt es sich durchaus herausfinden, ob sie „sich selbst akzeptieren"

Inhalt

Weiteres Vorgehen

- Die Unentscheidbarkeit von TM-DIAG ist erst der Anfang
- Unser Ziel ist, für interessantere Probleme zu zeigen, dass sie unentscheidbar sind

- Dafür werden wir als Zwischenschritt zunächst für zwei zu TM-DIAG ähnliche Probleme zeigen, dass sie unentscheidbar sind:
  - das Halteproblem für Turingmaschinen und
  - das Halteproblem für Turingmaschinen mit leerer Eingabe

Definition: TM-HALT

Gegeben: Turingmaschine M, Eingabe x für M

Frage: Hält M bei Eingabe x an?

Definition: TM-E-HALT

Gegeben: Turingmaschine M

Frage: Hält M bei Eingabe ϵ an?

Reduktionen

- Wir verwenden zum Nachweis der Unentscheidbarkeit zukünftig eine einfachere Methode als die „direkte Diagonalisierung“: **Reduktionen**

- Die Grundidee von Reduktionen ist, die Entscheidbarkeit eines Problems A auf die Entscheidbarkeit eines anderen Problems A' zurückzuführen

- Sie sollen uns Aussagen der folgenden Art ermöglichen:
  - wenn A' entscheidbar ist, dann ist auch A entscheidbar
- Daraus können wir dann folgern:
  - wenn A **nicht** entscheidbar ist, dann ist auch A' **nicht** entscheidbar

- Wir definieren hier Reduktionen formal für Sprachen und erlauben uns dann, die Definition auch auf andere algorithmische Entscheidungsprobleme zu übertragen

Definition

- Seien $L, L' \subseteq \Sigma^*$
- Eine totale, berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ heißt **Reduktion** von L auf L', wenn sie die folgende **Reduktionseigenschaft** hat:
  - für alle $x \in \Sigma^*$ gilt: $x \in L \iff f(x) \in L'$

- L heißt auf L' **reduzierbar**, falls es eine Reduktion von L auf L' gibt
  - Notation: $L \leqslant L'$

- Die Eigenschaft $x \in L \iff f(x) \in L'$ lässt sich auch anders (aber äquivalent) formulieren:
  - wenn $x \in L$ dann $f(x) \in L'$ und
  - wenn $x \notin L$ dann $f(x) \notin L'$

Reductionen: Erstes Beispiel (1/2)

- Wir werden Reduktionen auch auf der Ebene algorithmischer Entscheidungsprobleme verwenden:
  - Sind A, A' zwei solche Probleme, so schreiben wir $A \leqslant A'$, falls $L_A \leqslant L_{A'}$
- In Teil A der Vorlesung haben wir gesehen, dass sich das Nichtleerheitsproblem für endliche Automaten im Grunde wie das Erreichbarkeitsproblem für Graphen lösen lässt
- Diesen Zusammenhang präzisieren wir jetzt, indem wir zeigen, dass das Nichtleerheitsproblem auf das Erreichbarkeitsproblem reduzierbar ist

Definition: DFA-NonEmpty

Gegeben: DFA $\mathcal{A}$

Frage: Ist $L(\mathcal{A}) \neq \varnothing$?

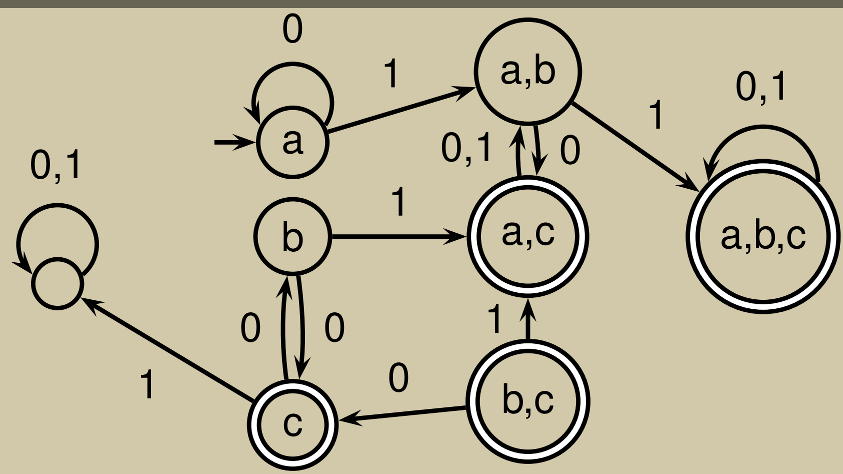
Beispiel

- Wir definieren eine Reduktionsfunktion, um zu zeigen, dass DFA-NonEmpty $\leqslant$ Reach gilt:
  - Für $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ sei $f(\mathcal{A}) \stackrel{\text{def}}{=} (G_\mathcal{A}, s, t)$, wobei:
    * $G_\mathcal{A} \stackrel{\text{def}}{=} (V_\mathcal{A}, E_\mathcal{A})$
    * $V_\mathcal{A} \stackrel{\text{def}}{=} Q \cup \{s, t\}$
    * $E_\mathcal{A} \stackrel{\text{def}}{=}$ $\{(s, q_0)\} \cup \{(q, t) \mid q \in F\} \cup \{(q, q') \mid \delta(q, \sigma) = q', \sigma \in \Sigma\}$

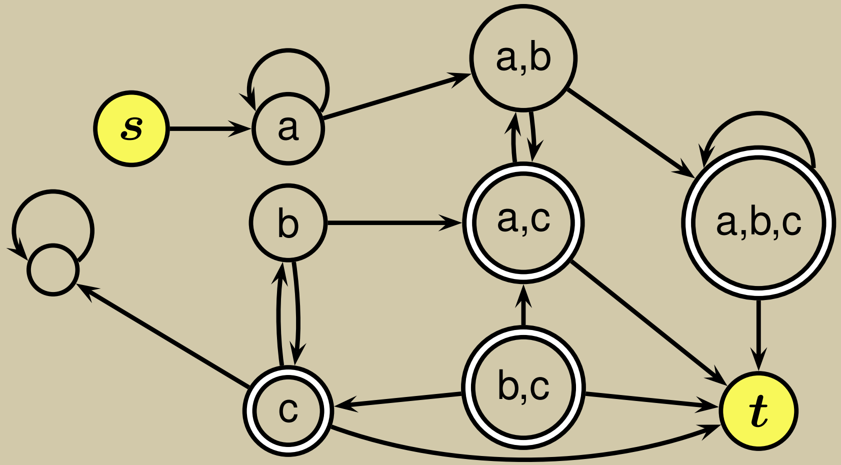
- Dann gilt:
  $\mathcal{A} \in$ DFA-NonEmpty $\iff f(\mathcal{A}) \in$ Reach
- Und natürlich ist f berechenbar

Reduktionen: Erstes Beispiel (2/2)

Beispiel: L_1 (Automaten)



Beispiel: L_2 (Graphen)



Reduktionen: Zweites Beispiel (1/2)

Satz 14.2

- PCP $\leqslant$ CFG-SCHNITT

Beweisskizze

- Sei $(u_1, v_1), \ldots, (u_k, v_k)$ eine Eingabe für PCP
  - (OBdA: $\$ \notin \Sigma$)

- Idee: Wir konstruieren Grammatiken G_1 und G_2 so, dass gilt:
  - $L(G_1)$ enthält alle Strings der Form

    $u_{i_1} \cdots u_{i_n} \$ i_n \cdots i_1$ mit $n \geqslant 1$
  - $L(G_2)$ enthält alle Strings der Form

    $v_{i_1} \cdots v_{i_n} \$ i_n \cdots i_1$ mit $n \geqslant 1$

- G_1: $S_1 \to u_1 S_1 1 \mid \cdots \mid u_k S_1 k \mid u_1 \$ 1 \mid \cdots \mid u_k \$ k$
- G_2: $S_2 \to v_1 S_2 1 \mid \cdots \mid v_k S_2 k \mid v_1 \$ 1 \mid \cdots \mid v_k \$ k$

- Dann sind äquivalent:
  - $(u_1, v_1), \ldots, (u_k, v_k)$ hat eine PCP-Lösung
  - $L(G_1) \cap L(G_2) \neq \varnothing$

Reduktionen: Zweites Beispiel (2/2)

Beispiel

- Steintypen: $\left[\frac{a}{aba}\right]$, $\left[\frac{ab}{bb}\right]$, $\left[\frac{baa}{aa}\right]$
- G_1:
  - $S_1 \to aS_11 \mid abS_12 \mid baaS_13 \mid a\$1 \mid ab\$2 \mid baa\$3$
- G_2:
  - $S_2 \to abaS_21 \mid bbS_22 \mid aaS_23 \mid aba\$1 \mid bb\$2 \mid aa\$3$
- Mögliche Lösung: $\left[\frac{a}{aba}\right]\left[\frac{baa}{aa}\right]\left[\frac{ab}{bb}\right]\left[\frac{baa}{aa}\right]$
- Zugehöriger String in $L(G_1) \cap L(G_2)$: $abaaabbaa\$3231$

Bemerkung

- Bei beiden Beispielen ist f formal nur für Strings definiert, die „vernünftige" Eingaben für DFA-NonEmpty bzw. PCP kodieren
- Die Funktion f kann jedoch zu einer totalen Funktion erweitert werden 14.1

Reduktionen und unentscheidbare Probleme

- Informelle Interpretation von Reduktionen:
  - Aus $A \leqslant A'$ folgt:
    * Falls es ein „Unterprogramm“ für A' gibt, so auch ein Programm für A
  - Falls $A \leqslant A'$ gilt, ist also in einem gewissen Sinne A nicht schwieriger als A'

Lemma 14.3

- Sind L, L' Sprachen mit $L \leqslant L'$, so gilt:

  (a) Ist L' entscheidbar, dann auch L

  (b) Ist L unentscheidbar, dann auch L'

- Um zu beweisen, dass ein Entscheidungsproblem A' unentscheidbar ist, genügt es also für ein schon als unentscheidbar bekanntes Problem A zu zeigen: $A \leqslant A'$

✎ Vorsicht, sprachliche Fehlerquelle: wir führen die Unentscheidbarkeit von A' auf die Unentscheidbarkeit von A zurück, indem wir zeigen, dass A auf A' reduzierbar ist!

Beweisidee

(a) Sei f eine Reduktion von L auf L'
  - Entscheidungsalgorithmus für L:
    * Bei Eingabe w, berechne $f(w)$
    * Teste $f(w) \in L'$ mit Hilfe eines Entscheidungsalgorithmus für L'
    * Akzeptiere, falls ja, lehne ab, falls nein

(b) Kontraposition von (a)

- Wir werden (unter anderem) zeigen:
  - TM-DIAG $\leqslant$ TM-HALT $\leqslant$ TM-E-HALT $\leqslant$ PCP
- Durch mehrfache Anwendung von Lemma 14.3 und mit PCP $\leqslant$ CFG-SCHNITT folgt dann, dass CFG-SCHNITT unentscheidbar ist

Weitere unentscheidbare Probleme (1/2)

Satz 14.4

- TM-HALT ist nicht entscheidbar

Beweisskizze

- Wir zeigen:
  $$\text{TM-DIAG} \leqslant \text{TM-HALT}$$
  – Dann folgt die Behauptung mit Lemma 14.3

- Prinzipielle Idee:
  $$M \mapsto (M, \text{enc}(M))$$

- Komplikation: M könnte bei Eingabe enc(M) anhalten und *ablehnen*
- Dann wäre
  $M \notin$ TM-DIAG aber
  $(M, \text{enc}(M)) \in$ TM-HALT
- Deshalb modifizieren wir die TM M so, dass sie nie anhält und ablehnt

Beweisskizze (Forts.)

- Für eine TM M sei M' die TM, in der alle Transitionen $\delta(q,\sigma) = (\text{nein}, \tau, d)$ durch Transitionen $\delta(q,\sigma) = (q,\sigma,\downarrow)$ ersetzt werden
- Dadurch wird erreicht, dass
  – M' anhält und akzeptiert, falls M akzeptiert, und
  – M' nicht anhält, falls M ablehnt oder nicht anhält

- Wir definieren die Funktion f durch:
  $$f(M) \stackrel{\text{def}}{=} (M', \text{enc}(M))$$
- Dann gilt:
  $$\begin{aligned} & M \in \text{TM-DIAG} \\ \iff & M \text{ akzeptiert enc}(M) \\ \iff & M' \text{ hält bei Eingabe enc}(M) \text{ an} \\ \iff & f(M) \in \text{TM-HALT} \end{aligned}$$

Weitere unentscheidbare Probleme (2/2)

Satz 14.5

- TM-E-HALT ist nicht entscheidbar

Beweisskizze

- Wir zeigen: TM-HALT $\leqslant$ TM-E-HALT

- Für jede TM M und jeden String $x \in \Sigma^*$ sei $M_{M,x}$ die TM, die
  - ihre eigentliche Eingabe löscht,
  - stattdessen x auf ihren String schreibt,
  - und dann M bei Eingabe x simuliert
- f sei definiert durch:

$$f((M,x)) \stackrel{\text{def}}{=} M_{M,x}$$

Beweisskizze (Forts.)

- f ist eine Reduktion von TM-HALT auf TM-E-HALT:
  - f ist total und berechenbar ✓
  - Es gilt:

$$\begin{aligned}
&(M,x) \in \text{TM-HALT} \\
\iff\ & M \text{ hält bei Eingabe } x \\
\iff\ & M_{M,x} \text{ hält bei Eingabe } \epsilon \\
\iff\ & M_{M,x} \in \text{TM-E-HALT} \\
\iff\ & f((M,x)) \in \text{TM-E-HALT}
\end{aligned}$$

Zusammenfassung

- Wir haben die Begriffe **entscheidbar** und **unentscheidbar** definiert
- Auf ähnliche Weise, wie wir uns von der algorithmischen Unlösbarkeit des „hello, world“-Problems überzeugt haben, lässt sich zeigen, dass das **Halte-Problem für Turingmaschinen unentscheidbar** ist
- Für viele andere Probleme lässt sich die Unentscheidbarkeit mit Hilfe von **Reduktionen** beweisen

Erläuterungen

Bemerkung 14.1

- Wenn wir eine Reduktionsfunktion f von einem algorithmischen Problem A auf ein Problem A' angeben, geben wir $f(x)$ nur für syntaktisch korrekte Eingaben x für A an
  - Wenn A einen Graphen als Eingabe „erwartet“, werden wir $f(G)$ also nur für Graphen G definieren
- Daraus können wir dann wie folgt eine totale Reduktionsfunktion $f' : \Sigma^* \to \Sigma^*$ gewinnen:
  - Für syntaktisch korrekte Eingaben $w = \text{enc}(G)$ ergibt sich dann $f'(w) \stackrel{\text{def}}{=} \text{enc}(f(G))$
  - Für Strings w, die keinen Graphen kodieren, setzen wir $f(w) \stackrel{\text{def}}{=} y$ für ein festes $y \notin L_{A'}$