Grundbegriffe der Theoretischen Informatik

Sommersemester 2018 - Thomas Schwentick

Teil D: Komplexitätstheorie

21: Zufallsbasierte Algorithmen

Version von: 10. Juli 2018 (12:59)

Einleitung

- In diesem Kapitel betrachten wir drei zufallsbasierte Algorithmen
  - Das sind Algorithmen, die Zufallsbits verwenden, um ein Problem zu lösen
  - Sie lösen ein Problem also nur mit einer gewissen Wahrscheinlichkeit und/oder ihre Laufzeit lässt sich nur mit einer gewissen Wahrscheinlichkeit beschränken

- Wir betrachten Algorithmen für
  - 3-SAT
  - das Problem, ob eine gegebene Zahl eine Primzahl ist
  - das Problem zu testen, ob ein arithmetischer Schaltkreis immer 0 ausgibt

Inhalt

Ein zufallsbasierter Algorithmus für 3-SAT

- Naiver Algorithmus für SAT: Probiere alle 2^n Belegungen der n Variablen der Eingabeformel φ aus
  $\rightarrow$ Laufzeit $\theta(|\varphi|2^n)$

- Hier betrachten wir einen einfachen zufallsbasierter Algorithmus für 3-SAT mit Laufzeit $\mathcal{O}(1.334^n)$

- Typisch für zufallsbasierte Algorithmen:
  - einfacher Algorithmus
  - komplizierte Analyse

Algorithmus 21.1 [Schöning]

Eingabe: φ mit Variablen $x_1, \ldots, x_n$

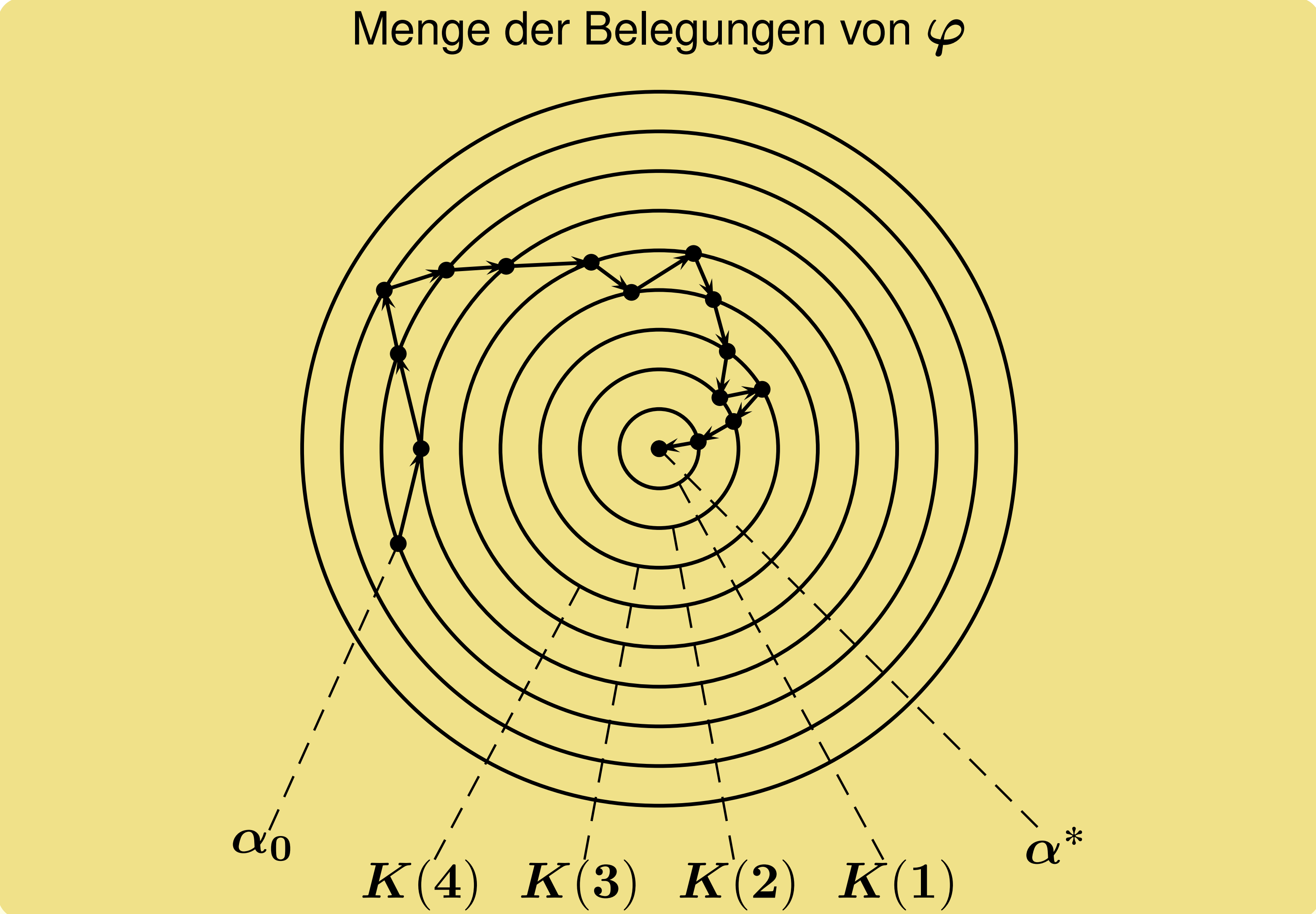
$\gamma(n) := \lceil 70\sqrt{n}(\frac{4}{3})^n \rceil$
for $\gamma(n)$ mal **do**
  Wähle zufällig eine Belegung α der Variablen
  for $3n$ mal **do**
    if $\alpha \models \varphi$ **then**
      Ausgabe „ja“, Fertig
    else
      Zufallsschritt
Ausgabe „nein“

Algorithmus Zufallsschritt

Wähle zufällig eine Klausel C, die von α nicht wahr gemacht wird
Wähle zufällig eine Variable x_k in C
Ändere α durch: $\alpha(x_k) := 1 - \alpha(x_k)$

- Wir nennen jeden der $\gamma(n)$ Durchläufe der äußeren Schleife einen *Versuch*

Schönings Algorithmus: Illustration



- α^*: Erfüllende Belegung von φ
- $K(j)$: Menge der Belegungen mit Hamming-Abstand j von α^*
- α_0: Zufällig gewählte erste Belegung

Ein zufallsbasierter Algorithmus für 3-SAT (Forts.)

Satz 21.2 [Schöning 99]

(a) Für erfüllbare Formeln findet Algorithmus 21.1 mit Wahrscheinlichkeit $> 0{,}9999$ eine erfüllende Belegung

(b) Für unerfüllbare Formeln gibt der Algorithmus „unerfüllbar“ aus

(c) Die Laufzeit ist $\mathcal{O}(|\varphi| n^{3/2} (\frac{4}{3})^n)$

Beweisskizze

- (b) und (c) sind klar
- Wir zeigen nun (a)

- Sei φ eine 3KNF-Formel und α^* eine erfüllende Belegung von φ
- Sei p die Wahrscheinlichkeit, dass der Algorithmus in einem einzelnen Versuch α^* findet (oder vorher auf eine andere erfüllende Belegung stößt)

- **Behauptung (A)**: $p \geqslant \frac{1}{7\sqrt{n}} (\frac{3}{4})^n$

Beweisskizze (Forts.)

- **Notation:**
  - $\alpha_0 \stackrel{\text{def}}{=}$ die in Zeile 3 eines Versuches gewählte Belegung
  - $K(j) \stackrel{\text{def}}{=}$ Menge aller Belegungen α mit $d(\alpha, \alpha^*) = j$

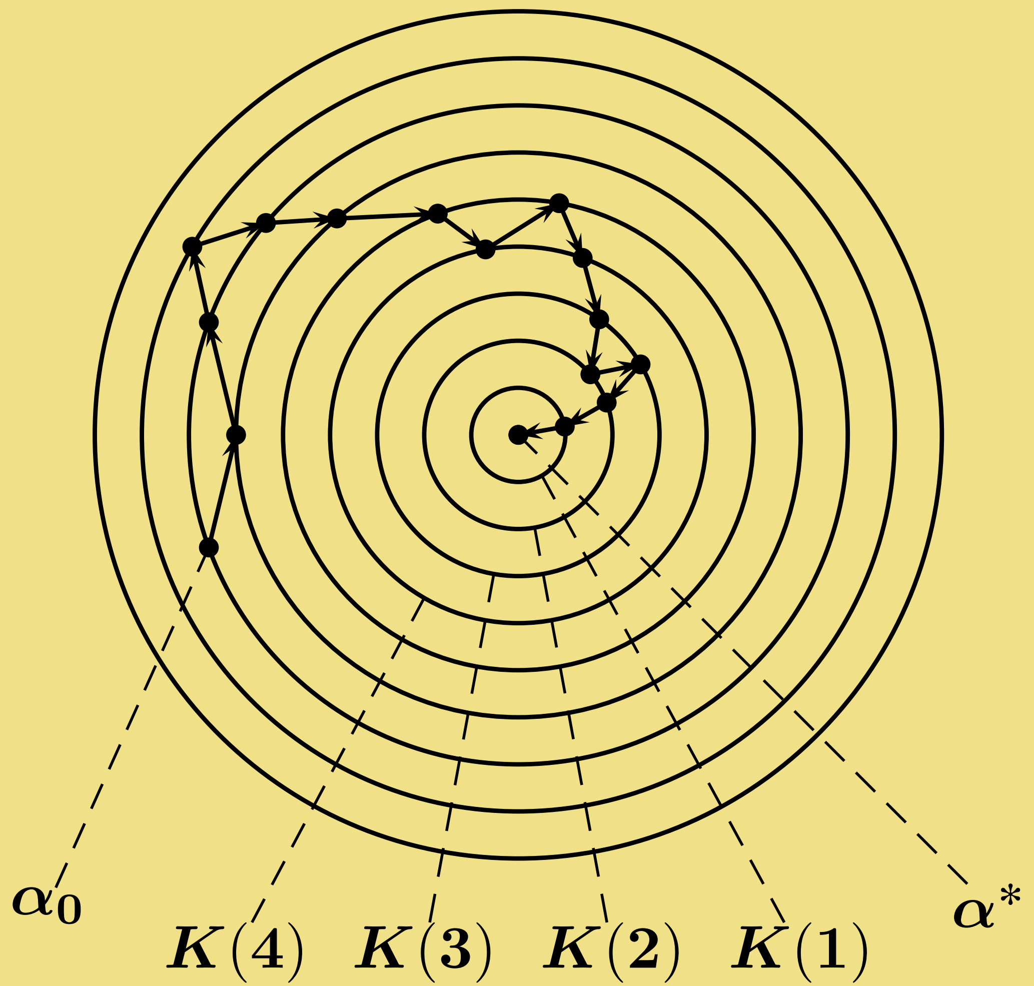
    ☞ $d(\cdot,\cdot)$: Hamming-Abstand
  - $p_j \stackrel{\text{def}}{=}$ W-keit, dass $\alpha_0 \in K(j)$ gilt
  - $q_j \stackrel{\text{def}}{=}$ W-Keit, dass α^* von einem $\alpha_0 \in K(j)$ in $\leqslant 3j$ Schritten erreicht wird oder der Algorithmus vorher eine andere Lösung findet

- Klar:
  - $p = \sum_{j=0}^{n} p_j q_j$
  - $p_j = \binom{n}{j} / 2^n$

- **Behauptung (B)**: $q_j \geqslant \frac{1}{7\sqrt{j}} (\frac{1}{2})^j$

Schönings Algorithmus: Illustration (Wdh.)

Menge der Belegungen von φ



- α^*: Erfüllende Belegung von φ
- $K(i)$: Menge der Belegungen mit Hamming-Abstand i von α^*
- α_0: Zufällig gewählte erste Belegung

Beweis von Behauptung (B) (1/2)

Beweisskizze

- Zur Erinnerung:
  - q_j: W-Keit, dass α^* von einem $\alpha_0 \in K(j)$ in $\leqslant 3j$ Schritten erreicht wird, oder der Alg. vorher eine andere Lösung findet
  - **Behauptung (B)**: $q_j \geqslant \frac{1}{7\sqrt{j}}\left(\frac{1}{2}\right)^j$

- Für $i \leqslant j$ sei $q_{j,i}$ die W-keit, dass der Alg. von $\alpha_0 \in K(j)$ aus in $2i + j$ Schritten α^* erreicht, oder vorher eine andere erfüllende Belegung erreicht

- Ein Weg, der in $2i + j$ Schritten von $\alpha_0 \in K(j)$ zu α^* führt, macht $i + j$ Schritte auf α^* zu und i Schritte von α^* weg
- Wir ordnen jedem solchen Weg einen String der Länge $2i + j$ über dem Alphabet $\{-, +\}$ zu ☞ $+$: auf α^* zu, $-$: von α^* weg

✎ Jedes Suffix des Strings muss dabei mindestens soviele $+$ wie $-$ enthalten

- Die Anzahl der Strings dieser Form ist $\binom{2i+j}{i}\frac{j}{2i+j}$

☞ ohne Beweis

Beweis von Behauptung (B) (2/2)

Beweisskizze (Forts.)

- Die W-Keit, in einem Schritt näher an α^* zu kommen, ist $\geqslant \frac{1}{3}$, denn:
  - Da C von α^* wahr gemacht wird, von dem jeweils aktuellen α aber nicht, gibt es mindestens eine Variable von C für die α und α^* sich unterscheiden
  - Wird die Belegung dieser Variablen in α geändert verringert sich der Abstand zu α^* um 1

➨ $q_{j,i} \geqslant \binom{2i+j}{i} \frac{j}{2i+j} \left(\frac{1}{3}\right)^{i+j} \left(\frac{2}{3}\right)^{i}$

➨
$$\begin{aligned} q_j &= \sum_{i=0}^{j} q_{j,i} \\ &\geqslant \frac{1}{3} \sum_{i=0}^{j} \binom{2i+j}{i} \left(\frac{1}{3}\right)^{i+j} \left(\frac{2}{3}\right)^{i} && ☞\ i \leqslant j \\ &\geqslant \frac{1}{2\sqrt{3\pi j}} \left(\frac{1}{2}\right)^{j} \geqslant \frac{1}{7\sqrt{j}} \left(\frac{1}{2}\right)^{j} && \Rightarrow \text{(B)} \end{aligned}$$

✎ Die letzte Zeile wird durch eine Reihe weiterer Umformungen erreicht, die unter anderem die Stirling-Approximations-Formel für die Fakultätsfunktion verwenden

Beweis von Satz 21.2

Beweisskizze

- Also:

$$p \geqslant \sum_{j=0}^n p_j q_j$$
$$\geqslant \sum_{j=0}^n \binom{n}{j}\left(\frac{1}{2}\right)^n \frac{1}{7\sqrt{j}}\left(\frac{1}{2}\right)^j$$
$$\geqslant \frac{1}{7\sqrt{n}}\left(\frac{1}{2}\right)^n \sum_{j=0}^n \binom{n}{j}\left(\frac{1}{2}\right)^j$$
$$= \frac{1}{7\sqrt{n}}\left(\frac{1}{2}\right)^n \left(1+\frac{1}{2}\right)^n \qquad \text{☞ Binomischer Lehrsatz}$$
$$= \frac{1}{7\sqrt{n}}\left(\frac{3}{4}\right)^n \qquad (=: \tilde{p})$$

➨ Behauptung (A)

- Die W-Keit, dass in einem Versuch weder α^* noch eine andere erfüllende Belegung gefunden wird ist also $\leqslant 1 - \tilde{p}$
- Die W-keit, dass dies $\gamma(n)$-mal passiert ist

$$\leqslant (1-\tilde{p})^{\gamma(n)}$$
$$\vdots$$
$$\leqslant e^{-10} < 5 \cdot 10^{-5}$$

➨ (a)

SAT-Solving

- Relativ effiziente Algorithmen für das SAT-Problem zu finden ist ein sehr wichtiges Forschungsthema

- Denn: viele andere algorithmischen Probleme lassen sich leicht auf die Erfüllbarkeit einer aussagenlogischen Formel zurückführen
- Anders gesagt: Reduktionen auf SAT sind häufig einfach

- Durch verfeinerte Techniken werden die Laufzeitschranken für SAT-Algorithmen kontinuierlich verbessert

- Einer der aktuell besten deterministischen Algorithmen für 3-SAT ist durch **Derandomisierung** von Schönings Algorithmus entstanden [Moser, Scheder 2011]
- Seine Laufzeit ist $\mathcal{O}(1.334^n)$
- Noch besser: $\mathcal{O}(1.3303^n)$ [Makino, Tamaki, Yamamoto 2011]

- Der beste zufallsbasierte Algorithmus hat Laufzeit $\mathcal{O}(1.30704^n)$ [Hertli 2011]

- Für praktische Anwendungen werden SAT-Solver heute in der Industrie „Routine-mäßig“ eingesetzt und haben für „typische“ Formeln eine gute Laufzeit

Inhalt

Primzahltests: Vorbereitung (1/2)

- Zur Erinnerung: eine *Primzahl* ist eine Zahl $p \in \mathbb{N}$, die außer 1 und p keine Teiler hat

Notation

- $\underline{k \mid n} \overset{\text{def}}{\Leftrightarrow} k$ ist Teiler von n, d.h.: es gibt ein $l \in \mathbb{N}$ mit $kl = n$ ☞ $3 \mid 9$, $4 \nmid 9$, $6 \nmid 9$
- $\underline{\text{ggT}(n, m)} \overset{\text{def}}{=}$ größter gemeinsamer Teiler von n und m, also die größte Zahl k mit $k \mid n$ und $k \mid m$ ☞ $\text{ggT}(6, 9) = 3$, $\text{ggT}(5, 7) = 1$
- $\underline{m \bmod n} \overset{\text{def}}{=}$ Rest bei Division von m durch n, also die eindeutig bestimmte Zahl k mit:
  (1) $n \mid (m - k)$ und
  (2) $0 \leqslant k < n$ ☞ $49 \bmod 9 = 4$
- $\underline{a \equiv_n b} \overset{\text{def}}{\Leftrightarrow} a$ und b haben den selben Rest bei Division durch n, also $a \bmod n = b \bmod n$ ☞ $7 \equiv_3 4$

Primzahltests: Vorbereitung (2/2)

Definition (PRIMES)

Gegeben: Zahl N

Frage: Ist N eine Primzahl?

Definition (COMPOSITES)

Gegeben: Zahl N

Frage: Ist N eine zusammengesetzte Zahl?

- Klar: COMPOSITES $\in$ **NP**

- Zu beachten:
  - Die *Länge* n der Eingabe ist jeweils $\lceil \log_2 N \rceil$

- Naiver Algorithmus für PRIMES:
  - Für alle $k \leqslant \sqrt{N}$:
    * Teste ob $k \mid N$
    * Falls ja, ablehnen
  - Falls Test ok, für alle k: akzeptieren

- Aufwand, falls N Primzahl ist:
  $$\theta(\sqrt{N}) = \theta(2^{\frac{1}{2}\log_2 N}) = \theta(2^{\frac{n}{2}})$$
  Tests ☞ exponentieller Aufwand

- Das geht besser, z.B. mit Hilfe des Zufalls...

Fermats Primzahltest

Satz 21.3 [Kleiner Satz von Fermat]

- Ist N eine Primzahl, so gilt für alle a, $1 \leqslant a < N$:
$$a^{N-1} \equiv_N 1$$

Beweisidee

- Wenn N eine Primzahl ist, bilden die Zahlen $1, \ldots, N-1$ mit der Operation $(x, y) \mapsto xy$ mod N eine multiplikative Gruppe mit $N-1$ Elementen
- In einer endlichen Gruppe G gilt für jedes Element g: $g^{|G|} = 1$

- Was ist, wenn N keine Primzahl ist?
  - $1^8 \equiv_9 1$ $\qquad$ $2^8 \equiv_9 4$
  - $3^8 \equiv_9 0$ $\qquad$ $4^8 \equiv_9 7$
  - $5^8 \equiv_9 7$ $\qquad$ $6^8 \equiv_9 0$
  - $7^8 \equiv_9 4$ $\qquad$ $8^8 \equiv_9 1$
- Für 75% der möglichen Werte für a würde durch den Test „$a^{9-1} \equiv_9 1$?“ erkannt werden, dass 9 keine Primzahl ist

- Lässt sich daraus ein zufallsbasierter Primzahltest konstruieren?
  (1) Wähle zufällig a, $1 \leqslant a < N$
  (2) Falls $a^{N-1} \equiv_N 1$, akzeptiere

- Fehler-Wahrscheinlichkeiten für andere zusammengesetzte Zahlen N:

4	33%
6	20%
8	14,3%
10	11%
12	9,1%
14	7,7%
16	13,3%
18	6%
⋮	⋮
561	99,1%

➞ Wir brauchen einen verbesserten Ansatz

✎ 561 ist die kleinste *Carmichael-Zahl*:
- Sie ist zusammengesetzt: $3 \times 11 \times 17$
- Für zu 561 teilerfremde a gilt: $a^{560} \equiv_{561} 1$

Primzahltest von Solovay-Strassen: (1/3)

- Der Primzahltest von Solovay-Strassen verwendet das *Jakobi-Symbol* $\left(\frac{a}{b}\right)$

Definition (Quadratischer Rest modulo p)

- Eine ganze Zahl a ist **quadratischer Rest modulo p**, für eine Primzahl p, $p \nmid a$, falls es eine Zahl c gibt mit $c^2 \equiv_p a$
- Ist p eine Primzahl, so sei

$$\left(\frac{a}{p}\right) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{wenn } a \text{ quadratischer Rest modulo } p \text{ ist,} \\ 0 & \text{falls } p \mid a \\ -1 & \text{andernfalls} \end{cases}$$

- Ist $p_1^{n_1} \times \cdots \times p_k^{n_k}$ die Primfaktorzerlegung der Zahl b, so sei $\left(\frac{a}{b}\right) \stackrel{\text{def}}{=} \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{n_i}$

✎ Für Primzahlen p heißt $\left(\frac{a}{p}\right)$ auch Legendre-Symbol

- Grundlage für den Solovay-Strassen-Test:

Proposition 21.4

- $a^{(N-1)/2} \equiv_N \left(\frac{a}{N}\right)$ gilt
  - für alle $a \in \{1, \ldots, N-1\}$, falls $N+2$ eine Primzahl ist;
  - für höchstens die Hälfte aller zu N teilerfremden $a \in \{1, \ldots, N-1\}$, falls $N+2$ *keine* Primzahl ist

- Daraus lässt sich direkt ein Primzahltest gewinen, der
  - Primzahlen immer erkennt
  - bei zusammengesetzten Zahlen eine Fehlerwahrscheinlichkeit $\leqslant \frac{1}{2}$ hat

- Es stellt sich allerdings die Frage, wie sich $\left(\frac{a}{N}\right)$ berechnen lässt, da die Definition die Primfaktorzerlegung von N verwendet...

Primzahltest von Solovay-Strassen: (2/3)

- Zur Berechnung von $\left(\frac{a}{N}\right)$ können wir die folgenden Regeln anwenden

1. $$\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right), \text{ falls } a > b$$
2. $$\left(\frac{a_1 a_2}{b}\right) = \left(\frac{a_1}{b}\right)\left(\frac{a_2}{b}\right)$$
3. $$\left(\frac{a}{b}\right) = (-1)^{(a-1)(b-1)/4}\left(\frac{b}{a}\right),$$ falls $a < b$ und a und b ungerade sind
4. $$\left(\frac{1}{b}\right) = 1$$
5. $$\left(\frac{2}{b}\right) = (-1)^{(b^2-1)/8}$$
6. $$\left(\frac{b-1}{b}\right) = (-1)^{(b-1)/2},$$ falls b ungerade

✎ Regel 3 wird auch quadratisches Reziprozitätsgesetz genannt

- Rekursive Berechung von $\left(\frac{a}{b}\right)$:
  - Falls ggT$(a, b) > 1$: Ergebnis $= 0$
  - Falls $a > b$: verwende Regel 1
  - Falls a oder b gerade: Spalte mit Regel 2 und der Definition Zweierpotenzen ab
  - Falls $a \in \{1, 2, b-1\}$: verwende die entsprechende Regel (4-6)
  - Andernfalls ($a < b$): verwende Regel 3

Beispiel

- $$\left(\frac{18}{11}\right) \stackrel{1.}{=} \left(\frac{7}{11}\right) \stackrel{3.}{=} (-1)^{6\cdot 10/4}\left(\frac{11}{7}\right) = -\left(\frac{11}{7}\right) \stackrel{1.}{=} -\left(\frac{4}{7}\right) \stackrel{2.}{=} -\left(\frac{2}{7}\right)^2 \stackrel{5.}{=} -1$$

- Bemerkung: Die Vorgehensweise ist also ähnlich zur Berechnung des ggT

Fakt

- $\left(\frac{a}{b}\right)$ lässt sich in Poly-Zeit in der Länge der Kodierung von a und b berechnen

Primzahltest von Solovay-Strassen (3/3)

Algorithmus 21.5 (Solovay-Strassen-Test)

Eingabe: N

1. Wähle zufällig $a < N$
2. Falls ggT$(a, N) > 1$ lehne ab
3. Akzeptiere, falls $a^{(N-1)/2} \equiv_N \left(\frac{a}{N}\right)$

- Eigenschaften des Solovay-Strassen-Tests:
  - Laufzeit: polynomiell in $\log_2 N$
  - Falls $N \in$ PRIMES: Wahrscheinlichkeit einer fehlerhaften Antwort: 0
  - Falls $N \notin$ PRIMES: Wahrscheinlichkeit einer fehlerhaften Antwort: $\leqslant \frac{1}{2}$

- Es gilt allerdings auch:

Satz 21.6 [Agrawal et al. 04]

- PRIMES $\in$ **P**

- Der zugehörige Polynomialzeit-Algorithmus ist jedoch ziemlich kompliziert
- Laufzeit: $\mathcal{O}((\log_2 N)^{10+\epsilon})$, inzwischen verbessert auf: $\mathcal{O}((\log_2 N)^{6+\epsilon})$

- Für praktische Zwecke ist der Solovay-Strassen-Test immer noch vorzuziehen

Fehler-Wahrscheinlichkeit zufallsbasierter Algorithmen (1/2)

Definition (Zufallsbasierter (f, g)-Algorithmus)

- Seien $f, g : \mathbb{N} \to \mathbb{R}$
- Wir sagen, dass ein Algorithmus A für eine Sprache L ein **zufallsbasierter (f, g)-Algorithmus** ist, falls er polynomielle Laufzeit hat und für jedes n gilt:
  - Für alle $w \in L$ der Länge n ist die W-keit einer fehlerhaften Antwort von A höchstens $f(n)$
  - Für alle $w \notin L$ der Länge n ist die W-keit einer fehlerhaften Antwort von A höchstens $g(n)$

- Der Solovay-Strassen-Test ist also
  - ein zufallsbasierter $(0, \frac{1}{2})$-Algorithmus mit polynomieller Laufzeit für PRIMES
  - ein zufallsbasierter $(\frac{1}{2}, 0)$-Algorithmus mit polynomieller Laufzeit für COMPOSITES
- Hier hängen f und g also nicht von n ab:
  - Für PRIMES ist $f(n) = 0$ und $g(n) = \frac{1}{2}$, für alle n

- Sind Algorithmen mit einer so großen Fehlerwahrscheinlichkeit relevant?
  - Ja, denn die Fehlerwahrscheinlichkeit lässt sich durch Wiederholung verkleinern

Fehler-Wahrscheinlichkeit zufallsbasierter Algorithmen (2/2)

- Was passiert, wenn wir den Solovay-Strassen-Test zweimal laufen lassen und N akzeptieren, wenn es den Test zweimal besteht?

- Falls $N \in$ PRIMES:

  Fehler-W-keit weiterhin 0

- Falls $N \notin$ PRIMES:
  - W-keit eines Fehlers im ersten Durchgang: $\leqslant \frac{1}{2}$
  - Also: für höchstens die Hälfte der Fälle wird der zweite Durchlauf erreicht
  - W-keit eines Fehlers im zweiten Durchgang: $\leqslant \frac{1}{2}$

  ➨ W-keit, dass insgesamt die falsche Antwort gegeben wird, ist $\leqslant \frac{1}{4}$

- Dieser Ansatz lässt sich weiterführen: wenn wir den Test k-mal ausführen, haben wir:
  - Falls $N \in$ PRIMES Fehler-W-keit 0
  - Falls $N \notin$ PRIMES Fehler-W-keit $\leqslant \frac{1}{2^k}$

- Für praktische Zwecke reicht es also sicherlich aus, den Solovay-Strassen-Test 100 mal zu wiederholen, um sich zu vergewissern, dass N eine Primzahl ist
  - Die Wahrscheinlichkeit, dass während der Berechnung ein Asteroid einschlägt, ist dann größer als die Irrtumswahrscheinlichkeit des Algorithmus

- Die beschriebene Vorgehensweise lässt sich für alle $(0, \frac{1}{2})$-Algorithmen und $(\frac{1}{2}, 0)$-Algorithmen anwenden

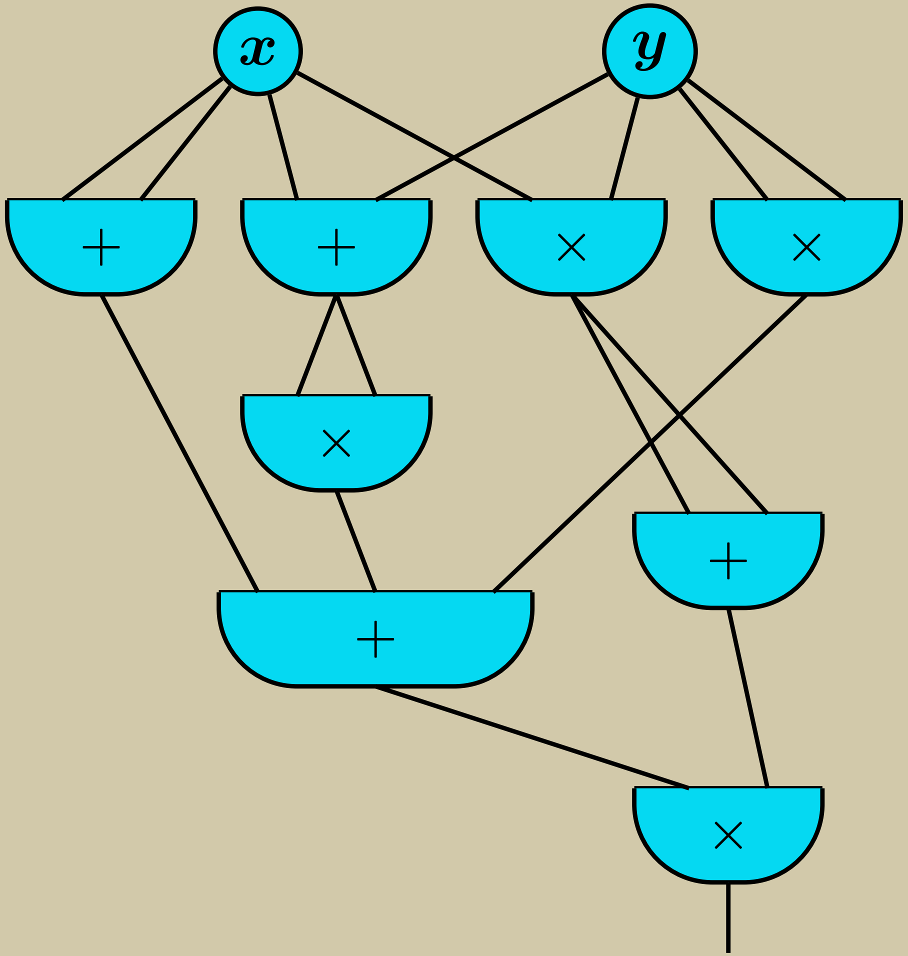
Inhalt

Zwischenbemerkung

- Wir haben zwei Beispiele für zufallsbasierte Algorithmen gesehen:
  - Schönings Algorithmus löst das **NP**-vollständige 3-SAT-Problem, allerdings nicht in polynomieller Zeit
  - Der Solovay-Strassen-Test löst das Primzahl-Problem in polynomieller Zeit, das ist aber auch ohne Zufall möglich
- Als drittes betrachten wir nun einen zufallsbasierten Algorithmus, der ein Problem in polynomieller Zeit entscheidet, für das kein deterministischer polynomieller Algorithmus bekannt ist

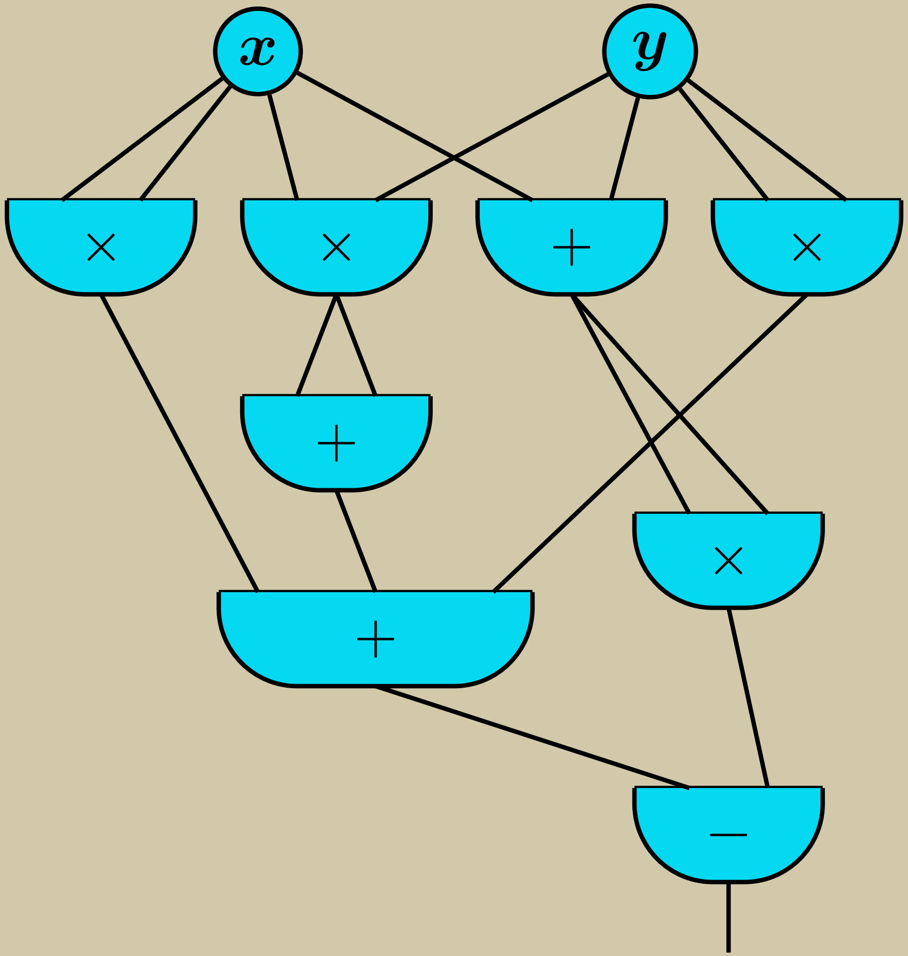
Arithmetische Schaltkreise

Beispiel



- Dieser *arithmetische Schaltkreis* berechnet das Polynom
  $$4x^2y + 2x^3y + 4x^2y^2 + 4xy^3$$

Beispiel (Forts.)



- Dieser arithmetische Schaltkreis berechnet das Polynom $\mathbf{0}$
- **Frage:** Wie lässt sich testen, ob ein arithmetischer Schaltkreis das Nullpolynom berechnet?

Nulltest für arithmetischer Schaltkreise (1/2)

Definition (ZeroCirc)

Gegeben: Arithmetischer Schaltkreis C

Frage: Berechnet C das Nullpolynom, hat es also für jede Eingabe die Ausgabe 0?

- ZeroCirc lässt sich natürlich dadurch lösen, dass das durch C repräsentierte Polynom p berechnet wird

- **Problem:** Die explizite Darstellung als Polynom kann exponentiell viele Terme haben
  - Deshalb führt dieser Ansatz nicht zu einem Polynomialzeit-Algorithmus
- Aber: Wir können ausnutzen, dass ein vom Nullpolynom verschiedenes Polynom für „die meisten“ Belegungen der Variablen nicht 0 ergibt

- Das folgende Lemma ist die Basis für einen zufallsbasierten Algorithmus für ZeroCirc

Lemma 21.7 [Schwartz-Zippel]

- Sei $p(x_1, \ldots, x_k)$ ein von 0 verschiedenes Polynom vom Grad $\leqslant d$ und S eine Menge ganzer Zahlen
- Werden $a_1, \ldots, a_k$ zufällig in S gewählt (gleichverteilt, unabhängig, mit Zurücklegen), so gilt:
  $$\Pr[p(a_1, \ldots, a_k) \neq 0] \geqslant 1 - \frac{d}{|S|}$$

- Im Prinzip können wir also wie folgt vorgehen:
  - Wähle $S \stackrel{\text{def}}{=} \{1, \ldots, 2d\}$
  - Wähle zufällig
    $$\vec{a} = (a_1, \ldots, a_k) \in S^k$$
  - Teste ob $p(\vec{a}) = 0$
  - Akzeptiere, falls dieser Test positiv verläuft
- Falls $p = 0$: Fehler-W-keit 0
- Falls $p \neq 0$: Fehler-W-keit $\leqslant \frac{1}{2}$

Nulltest arithmetischer Schaltkreise (2/2)

- Zwei Komplikationen:
  (1) Wir kennen den Grad d von p nicht
  (2) $p(\vec{a})$ kann *exponentiell lang* werden

- Lösung für (1): Hat $C \leqslant m$ Operationen, so ist der Grad von p höchstens 2^m
  → $S \stackrel{\text{def}}{=} \{1, \ldots, 2^{m+1}\}$

- Lösung für (2): Werte C modulo einer Zahl N aus

Algorithmus 21.8

Eingabe: C

1. Wähle $N \sim 2^{2m}$ zufällig
2. Wähle $S \stackrel{\text{def}}{=} \{1, \ldots, 2^{m+1}\}$
3. Wähle zufällig $\vec{a} \in S^k$
4. Teste ob $p(\vec{a})$ mod $N = 0$
5. Akzeptiere, falls Test positiv

- Klar: $p = 0 \Rightarrow$ der Test akzeptiert

- Es gilt auch: Falls $p \neq 0$ ist die W-keit, dass der Test *nicht akzeptiert* $\geqslant \frac{1}{20m}$
  - Denn: die W-keit, dass N eine Primzahl ist, die $p(\vec{a})$ nicht teilt, ist $\geqslant \frac{1}{10m}$ (ohne Beweis)

➡ Falls $p \neq 0$ ist die Fehler-W-keit $\leqslant 1 - \frac{1}{20m}$

- Durch $20m$ Wiederholungen lässt sich diese Fehler-W-keit auf $\frac{1}{2}$ senken
  (wegen $(1 - \frac{1}{x})^x \leqslant \frac{1}{e} \leqslant \frac{1}{2}$)

- Auf diese Art erhalten wir also einen polynomiellen zufallsbasierten $(0, \frac{1}{2})$-Algorithmus für ZeroCirc

Zusammenfassung

- Für manche algorithmischen Probleme sind die schnellsten bekannten zufallsbasierten Algorithmen schneller als die besten bekannten Algorithmen, die deterministisch arbeiten
  - Der beste bekannte zufallsbasierte Algorithmus für das 3-SAT-Problem ist beispielsweise deutlich schneller als der beste bekannte deterministische Algorithmus
- ZeroCirc ist ein Beispiel eines Problems, für das kein deterministischer Polynomialzeit-Algorithmus bekannt ist, aber zufallsbasierte Algorithmen, die in polynomieller Zeit laufen
- Häufig sind zufallsbasierte Algorithmen relativ einfach, ihre Analyse jedoch kompliziert
- Zufallsbasierte Algorithmen motivieren die Untersuchung von zufallsbasierten Komplexitätsklassen

☞ nächstes Kapitel

Literaturhinweise

- Manindra Agrawal, Neeral Kayal, and Nitin Saxena. Primes in P. *Annals of Mathematics*, 160(2):781–793, 2004
- Timon Hertli. 3-SAT faster and simpler - Unique-SAT bounds for PPSZ hold in general. In Rafail Ostrovsky, editor, *FOCS*, pages 277–284. IEEE, 2011
- Robin A. Moser and Dominik Scheder. A full derandomization of Schoening's k-SAT algorithm. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 245–252, New York, NY, USA, 2011. ACM
- Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. Derandomizing the HSSW algorithm for 3-SAT. *Algorithmica*, 67(2):112–124, 2013
- Christos M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994
- Uwe Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *FOCS*, pages 410–414, 1999