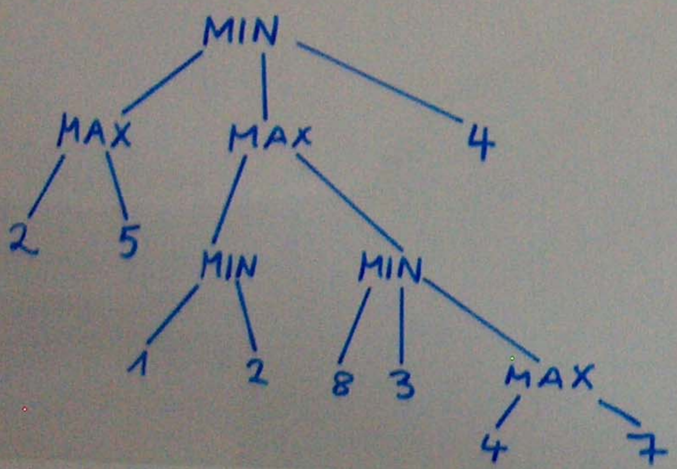
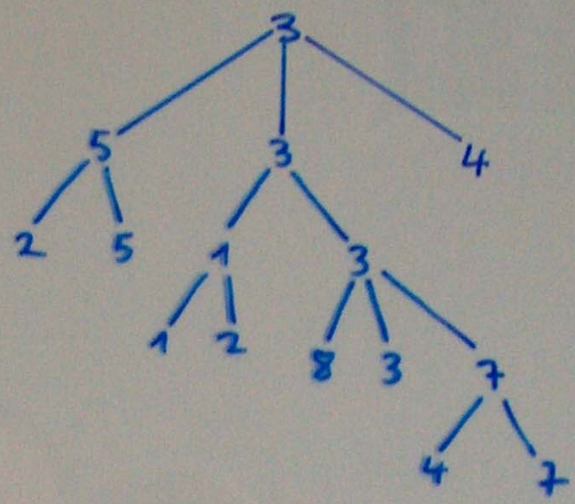
Kapitel 2. Spieltheoretische Methoden

- Ein **Spielbaum** ist ein gewurzelter Baum mit folgenden Eigenschaften:
  - Jedes Blatt ist mit einer reellen Zahl markiert
  - Knoten mit gerader (bzw. ungerader) Distanz von Wurzel sind mit MIN (bzw. MAX) markiert.
- Beispiel:



Auswertung des obigen Spielbaums



Hier: 2 Einschränkungen

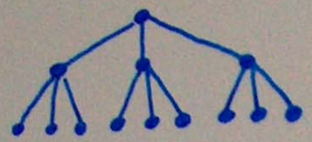
- Blätter sind mit 0 oder 1 beschriftet.

  ↪ MIN ≙ AND

  MAX ≙ OR → **AND-OR-Bäume**

- Sei $T_{d,k}$ der Baum mit:
  - Jedes Nicht-Blatt hat genau d Kinder

- Jedes Blatt hat Distanz $2k$ zur Wurzel

z.B. $T_{3,1}$ = 

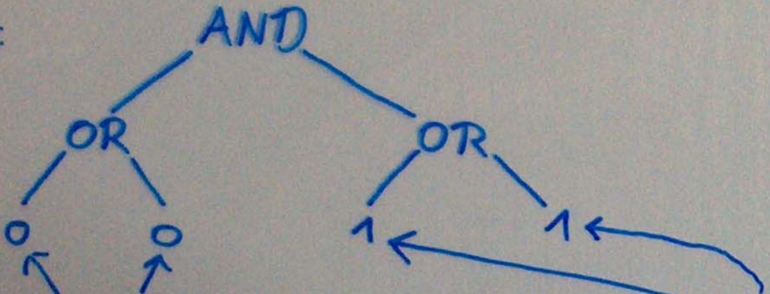
$\hookrightarrow T_{d,k}$ hat d^{2k} Blätter

<u>INPUT</u>: Baum $T_{d,k}$ und d^{2k} 0/1-Werte für die Blätter
(→ eindeutiger AND-OR-Baum)

<u>ZIEL</u>: Werte Wurzel vom $T_{d,k}$ aus.

Wir wollen dabei möglichst wenige der 0/1-Werte an den Blättern lesen.

<u>Beispiel</u>:



Nach Lesen dieser beiden Blätter müssen diese nicht mehr gelesen werden.

Man kann folgendes zeigen:

Sei A ein beliebiger deterministischer Auswertungsalgorithmus.
Dann gibt es eine 0/1-Beschriftung der Blätter vom $T_{d,k}$, bei der A alle d^{2k} Blätter lesen muß.

Im Folgenden: $d = 2 \rightarrow T_{2,k}$ hat 4^k Blätter

<u>Algorithmus rand Eval</u>

<u>eval(v)</u> % v ist ein Knoten vom $T_{2,k}$

if "v ist ein Blatt" then
  return Wert vom v
else
  Seien v_0 und v_1 die Kinder vom v
  Wähle zufällig ein $i \in \{0,1\}$
  $b := eval(v_i)$

```
if "v ist ein AND-Knoten" then
    if b = 0 then return(0)
    else return eval(v_{1-i})
    endif
  elseif % v ist also ein OR-Knoten
    if b = 1 then return(1)
    else return eval(v_{1-i})
    endif
  endif
endif
```

<u>Satz</u> Für jede 0/1-Beschriftung der Blätter vom $T_{2,k}$ ist der Erwartungswert für die Anzahl der gelesenen Blätter in einem Ablauf von randEval höchstens 3^k.

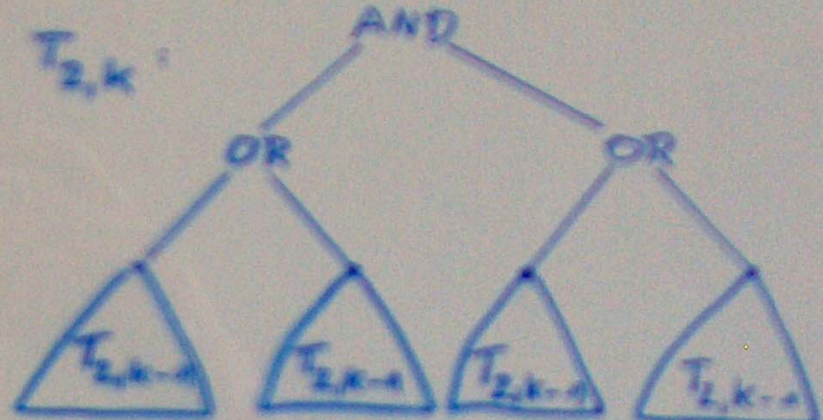
<u>Induktion über k</u>

<u>$k=0$</u> $T_{2,0}$: $\overset{v}{\bullet}$ (ein Blatt)

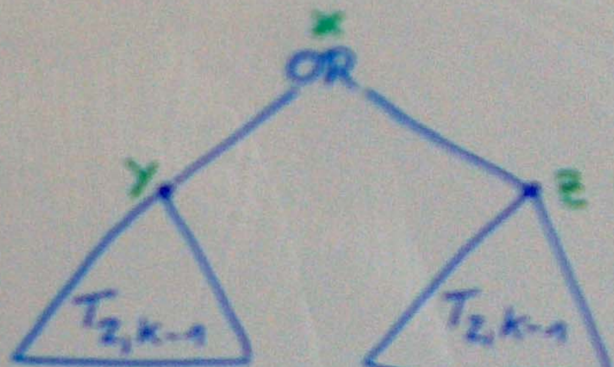
↳ Anzahl der auszuwertenden Blätter bei Aufruf $eval(v) = 1 = 3^0$

<u>$k>0$</u>:

$T_{2,k}$:



Betrachte zuerst den folgenden Baum



1. Fall: $eval(x) = 1$

$\rightarrow eval(y) = 1$ oder $eval(z) = 1$

Sei z.B. $eval(y) = 1$

Bei Aufruf $eval(x)$ wird mit Wahrscheinlichkeit $\frac{1}{2}$ $eval(y)$ als erstes aufgerufen.

Induktionshypothese:

$E[$Anzahl der ausgewerteten Blätter bei Aufruf $eval(v)]$
$\leq 3^{k-1}$ für $v \in \{y, z\}$

$\Rightarrow E[$Anzahl der ausgewerteten Blätter bei Aufruf $eval(x)]$

$$(*) \quad \leq \frac{1}{2} \cdot 3^{k-1} + \frac{1}{2} \cdot \underbrace{2 \cdot 3^{k-1}}_{\substack{\text{Kosten für den Fall} \\ eval(\cancel{y}) = 1,\ eval(z) = 0 \text{ und} \\ eval(z) \text{ wird als erstes aufgerufen}}} = \frac{3}{2} \cdot 3^{k-1}$$

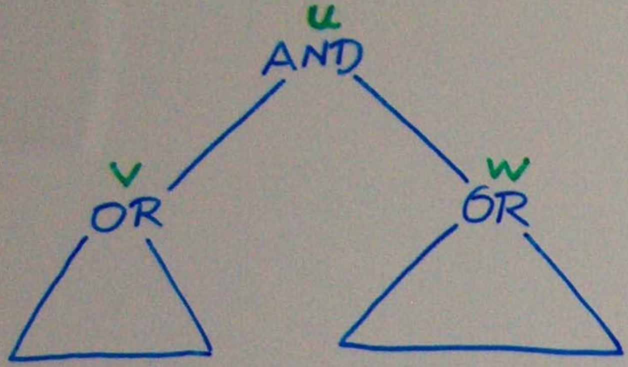
Bemerkung: Wir gehen hier von einem "Worst-Case-Input" aus: $eval(y) = 1$ und $eval(z) = 0$.
Der Fall $eval(y) = eval(z) = 1$ würde einen Erwartungswert für die Anzahl der ausgewerteten Blätter bei Aufruf $eval(x)$ von $3^{k-1} < \frac{3}{2} \cdot 3^{k-1}$ liefern.

2. Fall: $eval(x) = 0$

$\rightarrow eval(y) = eval(z) = 0$

$\rightarrow E[$Anzahl der ausgewerteten Blätter bei Aufruf $eval(x)]$
$\leq 2 \cdot 3^{k-1}$ $(**)$

Nun betrachten wir $T_{2,k}$



<u>1. Fall:</u> $eval(u) = 1$

$\rightarrow eval(v) = eval(w) = 1$

$\Rightarrow$ E[Anzahl der ausgewerteten Blätter bei Aufruf eval(u)]

$$\leq \underbrace{\frac{3}{2} \cdot 3^{k-1} + \frac{3}{2} \cdot 3^{k-1}}_{\text{Kosten für Auswertung von v und w nach (*)}} = 3^{k-1}$$

<u>2. Fall:</u> $eval(u) = 0$

$\rightarrow eval(v) = 0$ oder $eval(w) = 0$

Sei z.B. $eval(v) = 0$

Mit Wahrscheinlichkeit $\frac{1}{2}$ wird eval(v) als erstes aufgerufen.

E[Anzahl der ausgewerteten Blätter bei Aufruf eval(u)]

$$\leq \frac{1}{2} \cdot 2 \cdot 3^{k-1} \Big\} \text{ Kosten für Auswertung von v nach (**)}$$

$$+ \frac{1}{2} \Big(\underbrace{2 \cdot 3^{k-1}}_{\text{Kosten für Auswertung von v nach (**)}} + \underbrace{\frac{3}{2} \cdot 3^{k-1}}_{\text{Kosten für Auswertung von w, falls eval(w) = 1, nach (*)}} \Big)$$

$$= 3^{k-1} + 3^{k-1} + \frac{3}{4} \cdot 3^{k-1} < 3^k$$

$\square$

- randEval ist ein Las Vegas Algorithmus
- Im folgenden wollen wir eine untere Schranke für den Erwartungswert der Laufzeit eines beliebigen Las Vegas Algorithmus für ein Problem ableiten

Min Max-Prinzip, Spieltheorie

Beispiel: Stein-Schere-Papier
mit 2 Spielern: Roberta, Charles

Darstellung als Gewinn-Matrix:

Strategien vom Charles

Strategien vom Roberta	Schere	Papier	Stein
Schere	0	1	-1
Papier	-1	0	1
Stein	1	-1	0

Auszahlungsbetrag vom Charles an Roberta (Gewinn für Roberta)

- Stein-Schere-Papier ist ein **Null-Summen Spiel**:

Gewinn für Roberta
=
Verlust für Charles

- Jedes Null-Summen Spiel kann durch eine $(n \times m)$ Gewinn-Matrix M dargestellt werden.

n = Anzahl der Zeilen vom M
= Anzahl der Strategien für Roberta

m = Anzahl der Spalten vom M
= Anzahl der Strategien für Charles

$M_{i,j}$ = Gewinn für Roberta, falls sie Strategie i wählt und Charles Strategie j wählt.

- Angenommen Roberta wählt Strategie i

$\Rightarrow$ Sie wird mindestens Gewinn $\min\limits_j M_{ij}$ machen.

$\Rightarrow$ i ist optimale Strategie für Roberta, falls $\min\limits_j M_{ij}$ maximal.

Sei $V_R = \max\limits_i \min\limits_j M_{ij} =$

= Mindestgewinn für Roberta, falls sie optimal spielt.

- j ist optimale Strategie für Charles, falls $\max\limits_i M_{ij}$ minimal.

Sei $V_C = \min\limits_j \max\limits_i M_{ij} =$

= Maximalbetrag den Charles zahlen muß, falls er optimal spielt.

<u>Übung</u>: $\max\limits_i \min\limits_j M_{ij} \leq \min\limits_j \max\limits_i M_{ij}$

<u>Bemerkung</u>: Es kann

$\max\limits_i \min\limits_j M_{ij} < \min\limits_j \max\limits_i M_{ij}$

gelten.

Für Stein-Schere-Papier gilt z.B.

$V_R = \max\limits_i \min\limits_j M_{ij} = -1$

$V_C = \min\limits_j \max\limits_i M_{ij} = 1$

- Falls $V_R = V_C$, dann existiert ein Paar (p, γ) mit:

$$\min_j M_{p,j} = V_R = V_C = \max_i M_{i,\delta}$$

(p, γ) ist eine <u>reine Lösung</u> des Spiels oder ein <u>Sattelpunkt</u>

Beispiel

	1	2	3
1	0	1	2
2	-1	0	1
3	-2	-1	0

(1,1) ist ein Sattelpunkt für dieses Spiel.

Vorsicht: $M_{1,1} = M_{2,2} = M_{3,3}$ aber (2,2) oder (3,3) ist kein Sattelpunkt.

Randomisierte (gemischte) Strategien

- Eine randomisierte Strategie für Spieler $A \in \{$ Roberta, Charles $\}$ ist eine Wahrscheinlichkeitsverteilung auf den möglichen Strategien von A.

- rand. Strategie für Roberta:

  $p = (p_1, \ldots, p_n) \in [0,1]^n$ mit $\sum_{i=1}^{n} p_i = 1$

  $p_i = \text{Prob}[\text{Roberta wählt Strategie } i \text{ aus}]$

- rand. Strategie für Charles:

  $q = (q_1, \ldots, q_m) \in [0,1]^m$ mit $\sum_{j=1}^{m} q_j = 1$

  $q_j = \text{Prob}[\text{Charles wählt Strategie } j \text{ aus}]$

- Gewinn für Roberta (= Verlust für Charles) ist nun eine Zufallsvariable G

$$E[G] = \sum_{i=1}^{n} \sum_{j=1}^{m} p_i \cdot M_{ij} \cdot q_j$$

$$= p \cdot M \cdot q^T$$

Wieder sei $V_R = \max_p \min_q p \cdot M \cdot q^T$

$V_C = \min_q \max_p p \cdot M \cdot q^T$

- vom Neumann's MinMax Theorem

Für jedes 2-Spieler Null-Summen Spiel mit Gewinnmatrix M gilt:

$$\max_p \min_q p \cdot M q^T = \min_q \max_p p \cdot M \cdot q^T$$

- Ein Paar von rand. Strategien $(\hat{p}, \hat{q})$ mit $\min_q \hat{p} \cdot M \cdot q^T = V_R =$

$= V_C = \max_p p \cdot M \cdot \hat{q}^T$

heißt ein Sattelpunkt; $\hat{p}$ und $\hat{q}$ sind optimale gemischte Strategien (sind nicht unbedingt eindeutig).

- Beobachtung: Für eine feste Verteilung p (etwa $\hat{p}$) ist $p \cdot M \cdot q^T$ eine lineare Funktion $\alpha_1 \cdot q_1 + \alpha_2 \cdot q_2 + \dots + \alpha_m \cdot q_m$ mit $\alpha_i \in \mathbb{R}$.

Diese wird minimiert durch die Verteilung $q = (q_1, \dots, q_m)$ mit

$$q_i = \begin{cases} 1 & \text{falls } \alpha_i = \min\{\alpha_1, \dots, \alpha_m\} \\ 0 & \text{sonst} \end{cases}$$

Sei $e_k = (0, \dots, 0, 1, 0, \dots, 0)$

$\uparrow$ k-te Komponente

Also: $\max_p \min_q p \cdot M \cdot q^T =$

$= \max_p \min_j p \cdot M e_j^T$

Analog: $\min_q \max_p p \cdot M \cdot q^T =$

$= \min_q \max_i e_i \cdot M \cdot q^T$

Konsequenz (Loomis Theorem)

Für jedes 2-Spieler Null-Summen Spiel mit Gewinnmatrix M gilt:

$$\max_p \min_j p \cdot M \cdot e_j^T = \min_q \max_i e_i \cdot M \cdot q^T$$

Yaos Prinzip

- Sei P ein Berechnungsproblem (z.B. Auswerten von AND-OR-Bäumen)
- Fixiere eine endliche Menge $\mathcal{A}$ von deterministischen Algorithmen zur Lösung von P.
- Sei $\mathcal{I}$ die Menge aller Inputs der Länge n für P.

- Charles = Algorithmendesigner.
  - Eine (reine) Strategie für Charles ist ein deterministischer Algorithmus $A \in \mathcal{A}$.
  - Eine randomisierte Strategie für Charles ist eine Wahrscheinlichkeitsverteilung q über der Menge $\mathcal{A}$ von det. Algorithmen.

    Diese kann als ein Las Vegas Algorithmus A_q betrachtet werden.

- Roberta = Gegner (Adversary)
  - Eine (reine) Strategie für Roberta ist ein Input $I \in \mathcal{I}$.
  - Eine randomisierte Strategie für Roberta ist eine Wahrscheinlichkeitsverteilung p auf der Inputmenge $\mathcal{I}$.

    Sei I_p ein Zufallsinput, der gemäß der Verteilung p aus $\mathcal{I}$ ausgewählt wird.

$E[C(I_p, A_q)]$ = Erwartungswert

für die Laufzeit des Las Vegas Algorithmus A_q für einen Zufallsinput I_p

$$= \sum_{I \in \mathcal{I}} \sum_{A \in \mathcal{A}} \underbrace{p_I}_{\text{Prob[Roberta wählt Input I]}} \cdot \underbrace{C(I,A)}_{\text{Laufzeit des det. Alg. A bei Input I}} \cdot \underbrace{q_A}_{\text{Prob[Charles wählt Alg. A]}}$$

$$\Rightarrow \max_p \min_q E[C(I_p, A_q)]$$

(von Neumann)

$$= \min_q \max_p E[C(I_p, A_q)]$$

$$\max_p \min_{A \in \mathcal{A}} E[C(I_p, A)]$$

(Loomi)

$$= \min_q \max_{I \in \mathcal{I}} E[C(I, A_q)]$$

Konsequenz (Yaos Min Max Prinzip)

Für jede feste Inputverteilung p über $\mathcal{I}$ und jede feste Verteilung q über $\mathcal{A}$ ($\hat{=}$ Las Vegas Algorithmus) gilt:

$$\underbrace{\min_{A \in \mathcal{A}} E[C(I_p, A)]}_{\text{Erwartete Laufzeit eines optimalen det. Algorithmus für Inputverteilung } p} \leq \underbrace{\max_{I \in \mathcal{I}} E[C(I, A_q)]}_{\text{Wort case Laufzeit des Las Vegas Algorithmus } A_q}$$

Allgemeines Vorgehen:

- Sei P ein Berechnungsproblem

  Sei R ein beliebiger Las Vegas Algorithmus zur Lösung von P.

- R kann als eine Wahrscheinlichkeitsverteilung q über einer endlichen Menge $\mathcal{A}$ von det. Algorithmen angesehen werden (alle Münzen zu Beginn werfen).
- Wähle nun eine <u>geeignete</u> Verteilung p auf Inputs der Länge n.

$\overset{\text{Yao}}{\Longrightarrow}$ Es gibt einen Input I der Länge n mit:

$$E[\text{Laufzeit von } R \text{ auf Input } I] \geq$$

$$\min_{A \in \mathcal{A}} E[C(I_p, A)] \geq$$

$$\min_{A \in \mathcal{B}} E[C(I_p, A)]$$

$\uparrow$ Menge <u>aller</u> det. Algorithmen zur Lösung von P

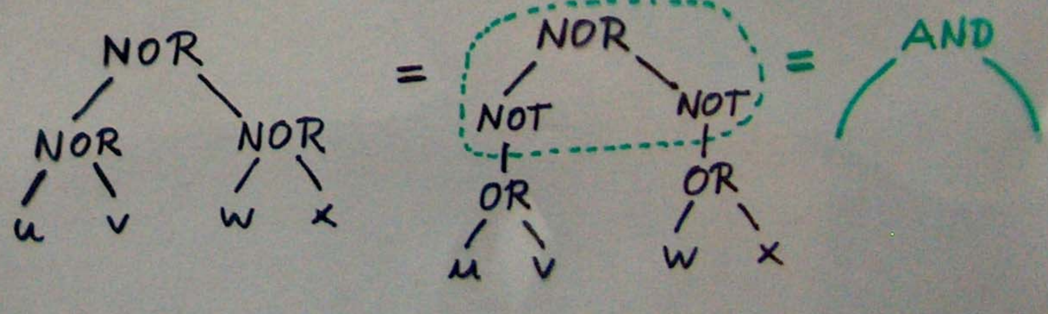
<u>Flexibilität von Yaos Methode</u>: Inputverteilung p kann frei gewählt werden.

Anwendung von Yaos MinMax Prinzip auf das Auswerten von AND-OR-Bäumen

<u>Beobachtung</u>:

Folgende 2 Bäume werten sich zum gleichen Wert an der Wurzel aus:

- $T_{2,k}$, wobei Knoten mit gerader (ungerader) Distanz zur Wurzel mit AND (OR) beschriftet sind
- $T_{2,k}$, wobei alle Knoten mit NOR (NOT-OR) beschriftet sind.

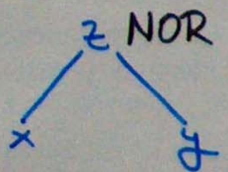


Im folgenden betrachten wir $T_{2,2k}$ wobei alle internen Knoten mit NOR beschriftet sind.

- Sei $\rho = \frac{3-\sqrt{5}}{2} \in [0,1]$

(*) Markiere jedes Blatt vom $T_{2,2k}$ unabhängig mit Wahrscheinlichkeit ρ mit 1 (und mit Wahrscheinlichkeit $1-\rho$ mit 0.

- Beobachtung: Betrachte folgenden Baum:



Setze x und y unabhängig voneinander mit Wahrscheinlichkeit ρ jeweils auf 1.

$$\Rightarrow \mathrm{Prob}[z=1] = \mathrm{Prob}[x=0 \wedge y=0]$$

$$= \mathrm{Prob}[x=0] \cdot \mathrm{Prob}[y=0]$$

$$= (1-\rho)^2 = \rho$$

$\Rightarrow$ Jeder Knoten in $T_{2,2k}$ wertet sich mit Wahrscheinlichkeit ρ zu 1 aus.

- Sei nun A ein beliebiger deterministischer Algorithmus zum Auswerten von NOR-Bäumen.

Wir "beweisen" eine untere Schranke für

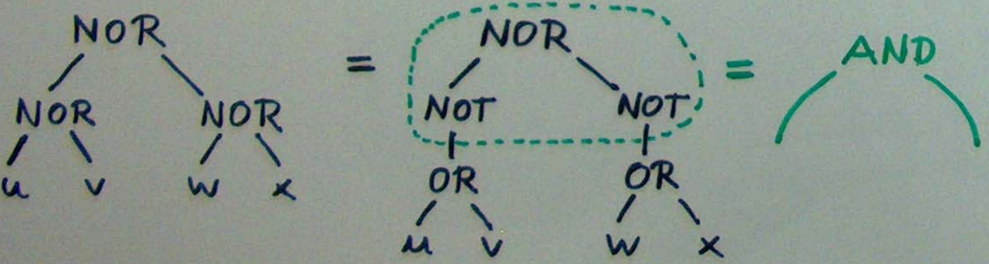
E[Laufzeit vom A, falls Input zufällig nach Methode (*) gewählt wird]

Anwendung von Yaos MinMax Prinzip auf das Auswerten von AND-OR-Bäumen

Beobachtung:

Folgende 2 Bäume werten sich zum gleichen Wert an der Wurzel aus:

- $T_{2,k}$, wobei Knoten mit gerader (ungerader) Distanz zur Wurzel mit AND (OR) beschriftet sind
- $T_{2,k}$, wobei alle Knoten mit NOR (NOT-OR) beschriftet sind.

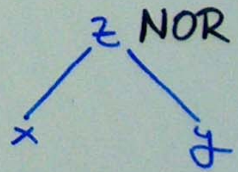


Im folgenden betrachten wir $T_{2,k}$ wobei alle internen Knoten mit NOR beschriftet sind.

- Sei $\rho = \frac{3-\sqrt{5}}{2} \in [0,1]$

(*) Markiere jedes Blatt von $T_{2,k}$ unabhängig mit Wahrscheinlichkeit ρ mit 1 (und mit Wahrscheinlichkeit $1-\rho$ mit 0.

- **Beobachtung:** Betrachte folgenden Baum:



Setze x und y unabhängig voneinander mit Wahrscheinlichkeit ρ jeweils auf 1.

$$\Rightarrow \text{Prob}[z=1] = \text{Prob}[x=0 \wedge y=0]$$
$$= \text{Prob}[x=0] \cdot \text{Prob}[y=0]$$

$$= (1-p)^2 = p$$

$\Rightarrow$ Jeder Knoten in $T_{2,k}$ wertet sich mit Wahrscheinlichkeit p zu 1 aus.

- Sei nun A ein beliebiger deterministischer Algorithmus zum Auswerten von NOR-Bäumen.

  Wir "beweisen" eine untere Schranke für

  E[Laufzeit von A, falls Input zufällig nach Methode (*) gewählt wird]

- Ein deterministischer Algorithmus A zum Auswerten von NOR-Bäumen ist ein depth-first pruning Algorithmus, falls für jeden Knoten v mit Kindern v_0, v_1 gilt: Es gibt ein $i \in \{0,1\}$, so dass A erst nur Blätter unterhalb vom v_i und danach nur Blätter unterhalb vom v_{1-i} besucht.

<u>Satz (ohne Beweis)</u> Für jeden deterministischen Algorithmus A zum Auswerten vom NOR-Bäumen gibt es einen depth-first pruning Algorithmus B mit:

E[Laufzeit von A, falls Input zufällig nach (*) gewählt wird] $\geq$
E[Laufzeit von B, falls Input zufällig nach (*) gewählt wird]

- Wir können uns also auf depth-first pruning Algorithmen beschränken

- Sei A ein depth-first pruning Algorithmus.

Sei $W(h) = E[$Anzahl der Blätter eines NOR-Baums der Tiefe h, die A auswertet, falls jedes Blatt mit Wahrscheinlichkeit p auf 1 gesetzt wird$]$

$$\Rightarrow W(h) = \underbrace{W(h-1)} + \underbrace{(1-p)} \cdot \underbrace{W(h-1)}$$

Aufwand zum Auswerten des "ersten" Kinds der Wurzel

Prob., dass ~~"zweite"~~ erste Kind der Wurzel sich zu 0 auswertet.

Aufwand zum Auswerten des "zweiten" Kinds der Wurzel

$$\Rightarrow W(h) = (2-p) \cdot W(h-1)$$

Da $W(0) = 1$, folgt $W(h) = (2-p)^h$

$h = \log_2(n)$

↑ Anzahl der Blätter.

$$\Rightarrow W(h) = (2-p)^{\log_2(n)}$$
$$= n^{\log_2(2-p)}$$
$$\geq n^{0{,}694}$$

<u>Satz</u>: Für jeden randomisierten Las Vegas Algorithmus R zum Auswerten vom AND-OR-Bäumen gibt es eine Belegung I der Blätter von $T_{2,k}$ mit 0/1-Werten, so dass:

$E[$Anzahl der Blätter vom $T_{2,k}$ die R bei Belegung der Blätter gemäß I auswertet$] \geq n^{0{,}694}$

wobei $n = 4^k$

Erinnerung: Für randEval war der Erwartungswert für die Anzahl der ausgewerteten Blätter vom $T_{2,k}$ bei jeder Belegung der Blätter mit 0/1-Werten $\leq 3^k$

$$n = 4^k \Rightarrow 3^k \leq n^{0{,}793}$$

Kann Yaos Methode so angewendet werden, dass wir eine untere Schranke vom 3^k erhalten?

Ja: Wahrscheinlichkeitsverteilung auf Inputmenge muß "cleverer" gewählt werden.

Derandomisierung und Nicht-Uniformität

- Ein Boolscher Schaltkreis mit n Inputs ist ein gerichteter azyklischer Graph mit den folgenden Eigenschaften:

(1) Es gibt genau n Knoten mit Eingangsgrad 0 (Inputgatter), diese sind mit $x_1, \ldots, x_n$ markiert.

(2) Es gibt genau einen Knoten mit Ausgangsgrad 0 (Ausgabegatter)

(3) Jeder Knoten, der kein Eingabegatter ist, ist mit AND, ODER, oder NOT markiert. Ist v mit NOT markiert, so hat v Eingangsgrad 1.

- Ein Boolscher Schaltkreis mit n Inputs berechnet auf natürliche Weise eine n-stellige Boolsche Funktion $f: \{0,1\}^n \to \{0,1\}$

- Ein randomisierter Boolscher Schaltkreis mit n Inputs hat zusätzlich zu den n Inputgattern noch m weitere Zufallsgatter $r_1, \ldots, r_m$ (m ist hier beliebig) mit Eingangsgrad 0.

  Ein randomisierter Boolscher Schaltkreis B mit n Inputs berechnet eine Funktion $f: \{0,1\}^n$, falls für jede Belegung von $x_1, \ldots, x_n$ mit 0/1-Werten gilt:

  - $f(x_1, \ldots, x_n) = 0 \Rightarrow B$ wertet sich zu 0 aus für jede Belegung der Zufallsgatter.
  - $f(x_1, \ldots, x_n) = 1 \Rightarrow B$ wertet sich zu 1 aus für mindestens die Hälfte aller Belegungen der Zufallsgatter

    d.h. Prob[B wertet sich bei zufälliger Belegung der Zufallsgatter zu 1 aus]

    $$\geq \frac{1}{2}$$

- Sei nun $f: \{0,1\}^* \to \{0,1\}$.

  Für $n \geq 0$ sei f_n die Einschränkung von f auf $\{0,1\}^n$, d.h.

  $f_n: \{0,1\}^n \to \{0,1\}$ und für alle $x \in \{0,1\}^n$: $f_n(x) = f(x)$.

- Sei $(B_n)_{n \geq 0}$ eine Folge von (randomisierten) Boolschen Schaltkreisen, wobei B_n n Inputgatter hat.

  $(B_n)_{n \geq 0}$ berechnet $f: \{0,1\}^* \to \{0,1\}$ falls für alle $n \geq 0$ gilt: B_n berechnet f_n.

- Die Folge $(B_n)_{n \geq 0}$ ist polynomiel, falls ein Polynom $p(n)$ existiert mit: Anzahl der Gatter von $B_n \leq p(n)$

Adlemans Theorem

Sei $f: \{0,1\}^* \to \{0,1\}$ eine Funktion, die von einer polynomiellen Folge von randomisierten Booleschen Schaltkreisen berechnet wird.
Dann wird f von einer polynomiellen Folge von Boolschen Schaltkreisen berechnet.

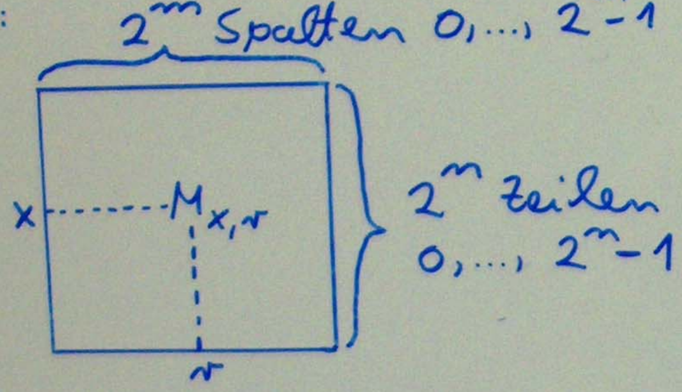
Beweis: Sei $(B_n)_{n \geq 0}$ eine polynomielle Folge von randomisierten Booleschen Schaltkreisen, die $f: \{0,1\}^* \to \{0,1\}$ berechnet.

Betrachte B_n: n Inputgatter $x_1, \ldots, x_n$
m Zufallsgatter $r_1, \ldots, r_m$

- Eine 0/1-Belegung der Inputgatter $x_1, \ldots, x_n$ (bzw. Zufallsgatter $r_1, \ldots, r_m$) wird mit einer Zahl $x \in \{0, \ldots, 2^n - 1\}$ (bzw. $r \in \{0, \ldots, 2^m - 1\}$) identifiziert

- Wir schreiben $B_n(x, r) = 1$, falls sich der Schaltkreis B_n unter diesen Belegungen zu 1 auswertet.

- Definiere Boolesche Matrix M wie folgt:

$M =$ 

$$M_{x,r} = \begin{cases} 1 & \text{falls } B_m(x,r) = 1 \\ 0 & \text{sonst} \end{cases}$$

- Lösche in M jede Zeile x mit $f(x) = 0$.

$\Rightarrow$ Danach sind in M in jeder Zeile mindestens die Hälfte aller Einträge 1.

$\Rightarrow$ Es existiert Spalte $r(1) \in \{0, \ldots, 2^m-1\}$, in der mindestens die Hälfte aller Einträge 1 ist.

- Sei $B_m(1)$ der Schaltkreis, der aus B_m entsteht, indem die Zufallsgatter $r_1, \ldots, r_m$ entsprechend der Belegung $r(1) \in \{0, \ldots, 2^m-1\}$ gesetzt werden.

- Lösche nun in M
  - Spalte $r(1)$
  - jede Zeile $x \in \{0, \ldots, 2^n-1\}$ mit $M_{x,r(1)} = 1$ (d.h. $B_m(x, r(1)) = 1$)

- Danach ist immer noch die Hälfte aller Einträge in jeder Zeile x von M gleich 1:

  Ist x eine Zeile, die nicht gelöscht wurde, so war $M_{x,r(1)} = 0$, aber vor dem Löschen war $M_{x,r}$ für mindestens die Hälfte aller r gleich 1

- Iteriere obigen Vorgang
  $\Rightarrow$ Belegungen $r(1), r(2), \ldots, r(k) \in \{0, \ldots, 2^m-1\}$

  (unrandomisierte) Schaltkreise $B_n(1), B_n(2), \ldots, B_n(k)$: $B_n(i)$ entsteht aus B_n, indem Zufallsgatter $r_1, \ldots, r_m$ entsprechend der Belegung $r(i)$ gesetzt werden.

- In jeder Phase wird die Anzahl der Zeilen von M mindestens halbiert.
  $\Rightarrow k \leq m$

Sei nun $B_n' = \underbrace{\bigvee_{i=1}^{k} B_n(i)}_{\text{poly. großer Schaltkreis}}$

$(B_n')_{n \geq 0}$ ist eine polynomielle Folge von (unrandomisierten) Schaltkreisen, die f berechnet. $\square$

<u>Bemerkung</u>: Der Beweis von Adlemans Theorem ist ein Beispiel für <u>Derandomisierung</u>: Wandle einen randomisierten Algorithmus in einen deterministischen Algorithmus um, wobei eine polynomielle Zeitschranke beibehalten wird.

Ist Derandomisierung immer möglich?

<u>Antwort</u>: Wahrscheinlich nicht.

Sei $\mathcal{C}$ eine Komplexitätsklasse (z.B. P, RP, NP).

Definiere die <u>nicht-uniforme</u> Version <u>$\mathcal{C}$/poly</u> von $\mathcal{C}$ wie folgt:

Eine Sprache $L \subseteq \{0,1\}^*$ gehört zur Klasse $\mathcal{C}/poly$, falls eine Funktion $\alpha: \mathbb{N} \to \{0,1\}^*$ existiert mit:

- Es existiert ein Polynom $p(n)$ mit $\forall n \geq 0: |\alpha(n)| \leq p(n)$
- $\{(x, \underbrace{\alpha(|x|)}_{\text{Advice-String für Inputs der Länge } n=|x|}) \mid x \in L\} \in \mathcal{C}$

Satz

- $L \in P/poly \Leftrightarrow$
  Es existiert eine polynomielle Folge von Booleschen Schaltkreisen, welche χ_L (charakteristische Funktion von L) berechnet.
- $L \in RP/poly \Leftrightarrow$
  Es existiert eine polynomielle Folge von randomisierten Booleschen Schaltkreisen, welche χ_L berechnet.

Beweisidee:

(1) Sei $(B_n)_{n \geq 0}$ eine polynomielle Folge von Booleschen Schaltkreisen, welche χ_L berechnet.

Definiere Advice-String $\alpha(n)$ wie folgt:

$\alpha(n)$ = binäre Kodierung des Schaltkreises B_n.

$\Rightarrow \{(x, \alpha(|x|)) \mid x \in L\} \in P$

$\Rightarrow L \in P/poly$

(2) Sei $L \in P/poly$

$\Rightarrow \{(x, \alpha(|x|)) \mid x \in L\} \in P$ wobei $\alpha(n) \leq p(n)$ für ein Polynom $p(n)$

Für jedes $n \geq 0$ kann ein polynomiell großer Schaltkreis B'_n mit $n + p(n)$ Inputgattern konstruiert werden, der alle Eingaben der Form $x\,\alpha(n)$ mit $|x| = n$, $x \in L$

akzeptiert (und sonst nichts akzeptiert)

Idee: Berechnungen von Turing-Maschinen können durch Boolesche Schaltkreise beschrieben werden (siehe Beweis für "SAT ist NP-vollständig)

Belege nun die letzten $\alpha(n)$ vielen Inputgatter von B_n' mit dem Advice-String $\alpha(n)$

$\Rightarrow$ Schaltkreis B_n

$(B_n)_{n\geq 0}$ ist eine polynomielle Folge von Booleschen Schaltkreisen, welche χ_L berechnet. $\square$

<u>Reformulierung von Adlemans Theorem</u>

$$P/poly = RP/poly$$

insbesondere: $RP \subseteq P/poly$

In einer "nichtuniformen Welt" kann also stets derandomisiert werden.

Andererseits wird vermutet, dass $P \subsetneq RP$ gilt.