# Informatik ᗡ: – Blatt 10

Rasmus Diederichsen

10. Juli 2014

## Aufgabe 10.1

### a)

$\mathcal{A}$ berechnet die Funktion $f(x) = 3^{3^x}$, für $x > 0$.

### b)

Das uniforme Kostenmaß ist nur sinnvoll, wenn primitive Operationen (Addition, Multiplikation) wirklich in $\mathcal{O}(1)$ durchgeführt werden können. Für reale Computer kann dies je nach Architektur nur für Zahlen $\leq 2^{32}$ oder $2^{64}$ passieren, da die Register keine größeren Zahlen fassen können.

Die Werte, die $y$ in $\mathcal{A}$ annimmt, laufen jedoch schon für relativ kleine $x$ ($x \lessapprox 3.366$ bei 64 Bit Integralzahlen) über diesen Wertebereich und sind dann nicht mehr in 32 oder 64 Bit darstellbar. Die arithmetischen Operationen sind dann nicht mehr atomar und der Aufwand wächst mit der Eingabegröße, ist also nicht mehr konstant.

### c)

Da der Wert von $y$ in jedem Schleifendurchlauf mit 3 potenziert wird, braucht Schleifendurchlauf $i$ $3(3^i \log 3)^2$ viele Schritte. Die Gesamtzahl an Schritten ist somit

$$f(x) = \sum_{i=1}^{x} 3(\log_2 3^{3^i})^2 = \sum_{i=1}^{x} 3(3^i \log_2 3)^2 \in \mathcal{O}\left(3^{2x}\right)$$

## Aufgabe 10.2

### a)

Das zugehörige Entscheidungsproblem ist

Gibt es ein $x \in \mathbb{R}^n$, sodass $c^T x \geq k$ mit $Ax \leq b$?

### b)

Optimierungsalgorithmus

```
def optimize(c, A, b):
   import numpy as np
   upperLim = np.sum(c) # groesser geht's nicht
   lowerLim = 0

   return binarySearch(0, upperLim)

def 𝒜(k):
      ... # entscheide ob max >= k ist mit c, A, b

   def binarySearch(lower, upper):
      if (upper == lower) return upper
      if (𝒜(upper)):
         return upper # wenn es ein Element >= upper gibt, muss es == upper sein
      if (not 𝒜(lower + 1)):
         return lower # wenn es kein Element > lower gibt, muss das maximum == lower sein
      if(𝒜(lower + (upper + lower) / 2)): # weiter in oberer Haelfte suchen
         return binarySearch(lower + (upper + lower) / 2, upper)
      else: # weiter in unterer Haelfte suchen
         return binarySearch(lower, upper - (upper + lower) / 2)
```

Im Algorithmus wird der Suchraum in jeder Iteration halbiert, es kann also maximal $\log_2 n$ viele rekursive Aufrufe geben, und auch das nur, falls $c = \{1\}^n$. Für jeden Aufruf von `binarySearch` wird einmal $\mathcal{A}$ aufgerufen, die Gesamtlaufzeit ist somit $\in \mathcal{O}\left(t_{\mathcal{A}}(n,m) \log_2 n\right)$. Ist man auch an dem zu dieser Lösung führenden $x$ interessiert, so errechnet man dies aus den $n$ Ungleichungen $Ax \leq b$, mithilfe von $c^T x = k$.

## Aufgabe 10.3

### a)

Man untersucht nacheinander alle Terme $T_i$ darauf, ob sie ein Literal und seine Negation enthalten. Falls nein, so kann man alle Literale so belegen, dass sie `true` sind und man terminiert, da nur ein $T_i$ `true` sein muss. Falls ja, so geht man weiter zur nächsten Klausel.

Ist auch der letzte Term unerfüllbar (der worst case), so hat man alle Literale nur einmal angeschaut, da es pro Literal nur eine Möglichkeit gibt, dass es `true` wird. Der Aufwand ist also maximal linear.

### b)

Das Theorem gälte, wenn die Überführung einer beliebigen Formel in DNF auch in polynomieller Zeit durchführbar wäre. Dieses Problem ist jedoch (anscheinend) bewiesenermaßen *NP*-schwer, und daher nur in $P$, wenn es auch in $NP$ ist und $P = NP$ gilt. Dies ist nicht bewiesen, daher gilt der Beweis nicht.

## Aufgabe 10.4

(1) $(x_{Stannis,1} \land x_{Sansa,1}) \lor (\neg x_{Stannis,1} \land \neg x_{Sansa,1})$

(2) $\bigwedge_t \bigvee_{v_1 \neq v_2} \neg(x_{v_1,t} \lor x_{v_2,t})$

→ Für jeden Zeitpunkt müssen mindestens zwei ungleiche Personen *nicht* in Tyrions Gemächern sein, über die anderen wird keine Aussage getroffen.

Es resultieren folgende Klauseln:

$$(\neg(x_{Stannis,1} \lor x_{Sansa,1}) \lor \neg(x_{Stannis,1} \lor x_{Shae,1}) \lor \neg(x_{Stannis,1} \lor x_{Sandor,1}) \lor \neg(x_{Sansa,1} \lor x_{Shae,1}) \lor \neg(x_{Sansa,1} \lor x_{Sandor,1}) \lor \neg(x_{Shae,1} \lor x_{Sandor,1})) \land (\neg(x_{Stannis,2} \lor x_{Sansa,2}) \lor \neg(x_{Stannis,2} \lor x_{Shae,2}) \lor \neg(x_{Stannis,2} \lor x_{Sandor,2}) \lor \neg(x_{Sansa,2} \lor x_{Shae,2}) \lor \neg(x_{Sansa,2} \lor x_{Sandor,2}) \lor \neg(x_{Shae,2} \lor x_{Sandor,2})) \land (\neg(x_{Stannis,3} \lor x_{Sansa,3}) \lor \neg(x_{Stannis,3} \lor x_{Shae,3}) \lor \neg(x_{Stannis,3} \lor x_{Sandor,3}) \lor \neg(x_{Sansa,3} \lor x_{Shae,3}) \lor \neg(x_{Sansa,3} \lor x_{Sandor,3}) \lor \neg(x_{Shae,3} \lor x_{Sandor,3}))$$

(3) $\bigwedge_{t_1 \neq t_2} \neg(x_{Stannis,t_1} \land x_{Stannis,t_2}) \land \neg(x_{Shae,t_1} \land x_{Shae,t_2})$

Es resultieren die Klauseln:

$$\neg(x_{Stannis,1} \land x_{Stannis,2}) \land \neg(x_{Shae,1} \land x_{Shae,2}) \land \neg(x_{Stannis,1} \land x_{Stannis,3}) \land \neg(x_{Shae,1} \land x_{Shae,3}) \land \neg(x_{Stannis,2} \land x_{Stannis,3}) \land \neg(x_{Shae,2} \land x_{Shae,3})$$

(4) $(x_{Sansa,1} \land x_{Sansa,2}) \lor \neg(x_{Sansa,1} \lor x_{Sansa,2}) \land \neg x_{Sansa,3}$

(5) $(x_{Sansa,2} \land x_{Shae,2}) \lor \neg(x_{Sansa,2} \lor x_{Shae,2})$

→ Sansa und Shae müssen sich entweder beide in Tyrions Gemächern aufgehalten haben, oder sie waren beide nicht dort. Zwar müssten sie dann *beide* am selben anderen Ort gewesen sein, dies spielt hier aber keine Rolle und ist nicht direkt darstellbar, da die einzigen Orte Tyrions Gemächer und Nicht-Tyrions-Gemächer sind.

(6) $x_{Shae,2}$

(7) $\bigwedge_t x_{Shae,t} \to y_t$. Dies lässt sich umformen zu $\bigwedge_t \neg x_{Shae,t} \lor y_t$. Es resultieren die Formeln

$$(\neg x_{Shae,1} \lor y_1) \land (\neg x_{Shae,2} \lor y_2) \land (\neg x_{Shae,3} \lor y_3)$$

(8*) $\bigwedge_{t_2 > t_1} \neg y_{t_1} \to \neg y_{t_2}$, was äquivalent ist zu $\bigwedge_{t_2 > t_1} y_{t_1} \lor \neg y_{t_2}$. Es resultieren die Klauseln

$$(y_1 \lor \neg y_2) \land (y_1 \lor \neg y_3) \land (y_2 \lor \neg y_3)$$

Wir sind nun in der Position, einige Schlussfolgerungen zu machen.

1. Wegen (6) wissen wir, dass $x_{Shae,2} =$ `true`.
2. Wegen (7) und 1. wissen wir, dass $y_2 =$ `true`.
3. Wegen (5) und 1. wissen wir, dass nicht nur $x_{Shae,2} =$ `true`, sondern auch $x_{Sansa,2} =$ `true`, da beide am gleichen Ort waren.
4. Wegen (4) und 3. wissen wir, dass $x_{Sansa,1} =$ `true`.
5. Wegen (1) und 4. wissen wir, dass $x_{Stannis,1} =$ `true`, da beide am gleichen Ort waren.
6. Wegen (3) und 5. wissen wir, dass $x_{Stannis,2} =$ `false`.
7. Wegen (2) und 3. wissen wir auch, dass $x_{Sandor,2} =$ `false` Alle Konjunkte müssen wahr sein, im zweiten Konjunkt sind jedoch wegen 3. alle Disjunkte `false`, die Shae und Sansa enthalten. Also muss $\neg(x_{Stannis,2} \vee x_{Sandor,2})$ `true` sein, mithin $x_{Stannis,2} =$ `false` und $x_{Sandor,2} =$ `false`.
8. Wegen (3) und 6. wissen wir, dass $x_{Stannis,3} =$ `false`.
9. Wegen (3) und 1. wissen wir, dass $x_{Shae,3} =$ `false`.
10. Wegen (4) wissen wir, dass $x_{Sansa,3} =$ `false`
11. Der einzige, der zur Stunde 3 bei Tyrion gewesen sein kann, war Sandor.

Der Mörder muss also Sandor Clegane sein.

## Aufgabe 10.5

## Aufgabe 10.6

Für $\chi =$ SAT ist die Lösung relativ simpel. Sei $n$ die Anzahl der vorkommenden Variablen. Die folgende Funktion gibt nach Eingabe einer Klauselmenge mithilfe von $\mathcal{A}_\chi$ einen Zeugen für $\mathcal{I}^*$ in Form einer Assoziationsliste zurück.

Hierbei sei `solvedp` ein Prädikat, dass bestimmt, ob ein Satz Formeln mit gegebenen Variablenzuweisungen bereits erfüllt ist, und `variables` retourniere die Liste von Variablen der SAT-Instanz.

Wer das hier lesen kann, bekommt einen Keks.

```
(defun witness (problem)
  (labels ((solve (formulas cur_assignments)
     (cond ((solvedp formulas cur_assignments) cur_assignments)
           ((not (𝒜χ formulas cur_assignments)) nil)
           (t (or (solve formulas
                         (cons (cons (find-if #'unassignedp
                                              (variables formulas)
                                                )
                                     t)
                               cur_assignments))
```

```
                    (solve formulas
                           (cons (cons (find-if #'unassignedp
                                                (variables formulas)
                                                    )
                                       nil)
                                 cur_assignments)))))))
        (solve problem nil)))
```