Prof. T. Nipkow
K. Kappelmann, J. Rädle, L. Stevens

KEINE ABGABE

Technische Universität München
Lehrstuhl für Logik und Verifikation

Einführung in die Theoretische Informatik

Sommersemester 2020 – Übungsblatt Lösungsskizze 12

AUFGABE 12.1. (*Wichtige Begriffe*) Stufe A

Überprüfen Sie, dass Sie die Folgenden Begriffe korrekt definieren können.

- Approximationsalgorithmus
- LOOP-berechenbar
- Primitiv-rekursiv
  - Projektionsfunktion
  - erweiterte Komposition
  - Cantorsche Paarungsfunktion

AUFGABE 12.2. (*SAT-Varianten*) Stufe C

Wir betrachten verschiedene Varianten von SAT, die auch NP-vollständig sind. Zeigen Sie diese NP-Vollständigkeit, indem Sie für jede Variante X eine Reduktion 3-KNF-SAT $\leq_p X$ angeben und $X \in \mathsf{NP}$ zeigen.

(a) Wir betrachten den ITE-Operator mit der Semantik $\texttt{ITE}(x, y, z) := (x \to y) \wedge (\neg x \to z)$. Eine ITE-Formel genügt der folgenden Grammatik:

$$F \to \texttt{ITE}(F, F, F) \mid x \mid \mathsf{true} \mid \mathsf{false} \qquad \text{für Variablen } x \in \mathcal{V}$$

ITE-SAT:
- Eingabe: Eine ITE-Formel F.
- Frage: Ist F erfüllbar?

(b) 3-OCC-KNF-SAT:
- Eingabe: Eine Formel F in KNF, bei der jede Variable höchstens dreimal auftritt.
- Frage: Ist F erfüllbar?

Lösungsskizze

(a)
- **NP-schwer:** Sei F eine Formel in 3-KNF. Wir wandeln F in polynomieller Zeit unter der Verwendung der folgenden semantischen Äquivalenzen in eine ITE-Formel:

$$\neg x \equiv \texttt{ITE}(x, \mathsf{false}, \mathsf{true}) \qquad x \wedge y \equiv \texttt{ITE}(x, y, \mathsf{false}) \qquad x \vee y \equiv \texttt{ITE}(x, \mathsf{true}, y)$$

Eine Klausel mit drei Literalen, z.B. $\neg x \vee \neg y \vee \neg z$ lässt sich damit umformen zu

$$\neg x \vee \neg y \vee \neg z \equiv \texttt{ITE}(\texttt{ITE}(x, \mathsf{false}, \mathsf{true}), \mathsf{true}, \texttt{ITE}(\texttt{ITE}(y, \mathsf{false}, \mathsf{true}), \mathsf{true}, \texttt{ITE}(z, \mathsf{false}, \mathsf{true})))$$

Jede Klausel wird in eine ITE-Formel übersetzt, welche nur um einen konstanten Faktor größer ist. Entsprechend übersetzt sich jede 3KNF-Formel in eine semantisch äquivalente ITE-Formel, die nur um einen konstanten Faktor größer ist. Die Korrektheit folgt aus der Verwendung semantisch-äquivalenten Umformungen.
- $\in$ **NP**: Eine erfüllende Belegung ist ein geeignetes Zertifikat, das es maximal genauso groß ist wie die Formel F und in polynomieller Zeit geprüft werden kann.

(b)
- **NP-schwer:** Sei F eine Formel in 3-KNF. Somit $F = \bigwedge_{i=1}^{n} K_i$ und $K_i = \bigvee_{j=1}^{m} L_{ij}$ mit $m \leq 3$. O.b.d.A. kommen keine Variablen doppelt in einer Klausel vor. Sei v die Anzahl der in F verwendeten Variablen. Dann ersetzten wir in jeder Klausel die Variablen mit für diese Klausel spezifische Variablen und fügen eine Bedingung ein, die erzwingt, dass alle Kopien einer Variable in der Eingabe den gleichen Wert zugewiesen bekommen. Diese Bedingung hat eine Ringstruktur, um Variablenverwendungen zu sparen: $(x_{k,1} \to x_{k,2}) \wedge (x_{k,2} \to x_{k,3}) \wedge \ldots (x_{k,n} \to x_{k,1})$.

$$f(F) = \bigwedge_{i=1}^{n} f(K_i) \wedge \bigwedge_{k=1}^{v} \bigwedge_{i=1}^{n} (\neg x_{k,i} \vee x_{k,(i \bmod n)+1})$$

$$f(K_i) = \bigvee_{j=1}^{m} f(L_{ij})$$

$$f(L_{ij}) = \begin{cases} x_{k,i} & \text{falls } L_{ij} = x_k \\ \neg x_{k,i} & \text{falls } L_{ij} = \neg x_k \end{cases}$$

Diese Reduktion ist in polynomieller Zeit berechenbar, da nur die Eingabe einmal kopiert werden muss und ein zusätzlicher Term der Größe $m \cdot v$ erzeugt wird.
Korrektheit: Aus jeder erfüllenden Belegung σ für F können wir eine erfüllende Belegung σ' für $f(F)$ konstruieren mit $\sigma'(x_{k,i}) = \sigma(x_k)$. Auch können wir aus jeder erfüllenden Belegung σ für $f(F)$ ein Belegung für F konstruieren. Da $\sigma(x_{k,i}) = \sigma(x_{k,j})$ für alle i, j gelten muss, können wir einfach σ' über $\sigma'(x_k) = \sigma(x_{k,1})$ definieren. Somit ist die Konstruktion erfüllbarkeitserhaltend und die Reduktion korrekt.

- $\in$ **NP**: Eine erfüllende Belegung ist ein geeignetes Zertifikat, das es maximal genauso groß ist wie die Formel F und in polynomieller Zeit geprüft werden kann.

AUFGABE 12.3. (*Nullstellen Problem*) Stufe C

Wir betrachten das Nullstellen-Problem:

Gegeben Ein Polynom p über Variablen $x_1, \ldots, x_n$ mit ganzzahligen Koeffizienten.

Problem Gibt es eine ganzzahlige Nullstelle, also $v_1, \ldots, v_n \in \mathsf{Z}$ mit $p(v_1, \ldots, v_n) = 0$?

(a) Zeigen Sie, dass das Nullstellen-Problem NP-hart ist, indem Sie eine Reduktion von SAT angeben. Hinweis: Die folgenden Beziehungen könnten dabei nützlich sein:

$$f(x)g(x) = 0 \iff f(x) = 0 \vee g(x) = 0$$
$$f(x)^2 + g(x)^2 = 0 \iff f(x) = 0 \wedge g(x) = 0$$

(b) Wo liegt das Problem bei folgendem "Beweis", dass Nullstellen in NP ist:

> Ein Zertifikat ist genau eine Nullstelle von p, also ein Vektor von Zahlen $v_1, \ldots, v_n \in \mathsf{Z}$. Ein Verifikator muss nun lediglich überprüfen, ob $p(v_1, \ldots, v_n) = 0$. Nach Satz 5.10 ist das Problem damit in NP.

Lösungsskizze

(a) Ausgehend von einer Booleschen Formel F konstruieren wir ein Polynom, welches genau dann eine Nullstelle hat, wenn F erfüllbar ist. Wie man $\wedge$ und $\vee$ mit Polynomen darstellt, ist schon vorgegeben, es fehlt lediglich die Negation. Wir können aber annehmen, dass die Negation in F lediglich auf Variablen vorkommt, da man jede Formel mit den Gesetzen von deMorgan leicht in diese Form bringen kann.
Aus jeder aussagenlogischen Variablen x_i wird nun eine ganzzahlige Variable X_i im Polynom. X_i soll genau dann null sein, wenn x_i false ist.
Wir führen einen Term ein, der dafür sorgt, dass jede Variable nur den Wert 0 oder 1 haben kann:

$$\sum_{1 \le i \le n} (X_i(1 - X_i))^2$$

Der Rest des Polynoms hat nun dieselbe Struktur wie die Formel. Wir stellen ein positives Literal durch $(1 - X_i)$ dar und ein negatives durch X_i. Eine Disjunktion wird zu einem Produkt, und die Konjunktion zu einer Summe der Quadrate.
Beispielsweise wird die Formel

$$(x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_3)$$

durch das Polynom

$$((1 - X_1)X_2)^2 + ((1 - X_2)(1 - X_3))^2 + \sum_{1 \le i \le 3} (X_i(1 - X_i))^2$$

dargestellt. Die Nullstellen des Polynoms entsprechen nun genau den erfüllenden Belegungen der Formel. Offensichtlich ist die Reduktion polynomiell.

(b) Das Problem ist, dass die ganzzahlige Nullstelle beliebig groß sein kann, und nicht polynomiell beschränkt in der Größe der Eingabe (des Polynoms). Damit kann der Verifikator nicht in Polynomialzeit laufen.
Anmerkung: Tatsächlich ist Nullstellen nicht in NP, denn es handelt sich hier um Hilberts 10. Problem (vgl. Satz 4.68), welches unentscheidbar ist.

AUFGABE 12.4. (*All You Need is Recursion*)

In den nachfolgenden Aufgaben dürfen Sie die erweiterte Komposition von primitiv rekursiven Funktionen verwenden.

(a) Zeigen Sie: Die Funktion

$$\mathit{ifthen}(n, a, b) = \begin{cases} a, & n \neq 0 \\ b, & n = 0 \end{cases}$$

ist primitiv-rekursiv.

(b) Zeigen Sie: Die Funktion

$$tower(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$$

die einen 2-er Turm der Höhe n berechnet, ist primitiv-rekursiv. Dabei soll $tower(0) = 1$ gelten.

(c) Für den Binomialkoeffizient gilt $\binom{n}{0} = 1$ und $\binom{0}{m} = 0$ für $n \in \mathbb{N}$ und $m \in \mathbb{N}_+$. Außerdem gilt für alle $n, m \in \mathbb{N}$ die Rekursionsgleichung:

$$\binom{n+1}{m+1} = \binom{n}{m+1} + \binom{n}{m}.$$

Für fixiertes $m \in \mathbb{N}$ betrachten wir nun die Funktion $b_m : \mathbb{N} \to \mathbb{N}, n \mapsto \binom{n}{m}$. Zeigen Sie: Für alle $m \in \mathbb{N}$ ist die Funktion b_m primitiv rekursiv.

Lösungsskizze

(a)

$$\begin{aligned} ifthen(0, a, b) &= b \\ ifthen(n+1, a, b) &= a \end{aligned}$$

(b) Die Multiplikation ist nach VL (Beispiel 5.37) primitiv-rekursiv. Wir definieren nun

$$\begin{aligned} twopow(0) &= 1 \\ twopow(n+1) &= mult(twopow(n), 2) \\ tower(0) &= 1 \\ tower(n+1) &= twopow(tower(n)) \end{aligned}$$

(c) Beweis per Induktion über $m \in \mathbb{N}$.
Fall $m = 0$: Dann ist $b_m(n)$ die konstante Eins-Funktion und somit primitiv-rekursiv (vgl. VL Beispiel 5.36).
Fall $m + 1$: Per IH ist b_m primitiv-rekursiv. Mit Hilfe des erweiterten Kompositionsschemas folgt dann, dass auch

$$\begin{aligned} b_{m+1}(0) &= 0 \\ b_{m+1}(n+1) &= b_{m+1}(n) + b_m(n) \end{aligned}$$

primitiv-rekursiv ist.