Prof. Javier Esparza
Philipp Czerner, Martin Helfrich

Technische Universität München
Lehrstuhl für Theoretische Informatik

Einführung in die Theoretische Informatik

Sommersemester 2021 – Hausaufgabenblatt 10

- Diese Woche werden alle Aufgaben korrigiert.
- Wenn Sie einen Beweis aufstellen, von dem Sie wissen, das einzelne Schritte problematisch oder unvollständig sind, merken Sie dies bitte in Ihrer Lösung an, damit wir das bei der Korrektur positiv berücksichtigen können.

Aufgabe H10.1. (*Auf- und Ab*) 0.5+0.5+0.5+0.5+0.5+0.5 Punkte

Wir betrachten folgende Eigenschaften: (1) abzählbar, (2) rekursiv aufzählbar, (3) semientscheidbar, und (4) entscheidbar. Geben Sie für jede der folgenden Mengen an, welche der vier Eigenschaften erfüllt sind. Bitte beachten Sie, dass (2-4) nur auf Sprachen definiert sind.

(a) $\emptyset$

(b) $\{w \in \mathbb{R} : w < \frac{1}{\sqrt{2}}\}$

(c) $\{w \in \{0, ..., 9\}^* : (0.w)_{10} < \frac{1}{\sqrt{2}}\}$

(d) $\{w \in \{0,1\}^* : \{1, 10, 111\} \cap L(M_w) \neq \emptyset\}$

(e) $\{w \in \{0,1\}^* \mid \forall v \in L(M_w) : (7(v)_2 - 31)^2 > 8\}$

(f) $\{w \in \{0,1\}^* : M_w$ hält für ein $v \in \{0,1\}^*$ in $|v|$ Schritten$\}$

Eine Begründung ist nicht erforderlich. Für (c) verwenden wir die Notation aus H5.6. Sie können folgende Tabelle verwenden:

	(1)	(2)	(3)	(4)
(a)	·	·	·	·
(b)	·	·	·	·
(c)	·	·	·	·
(d)	·	·	·	·
(e)	·	·	·	·
(f)	·	·	·	·

Lösungsskizze.

	(1)	(2)	(3)	(4)
(a)	×	×	×	×
(b)	·	·	·	·
(c)	×	×	×	×
(d)	×	×	×	·
(e)	×	×	×	×
(f)	×	×	×	·

Anmerkungen: (2) und (3) sind äquivalent. (c) Die Sprache kann man entscheiden, indem man die Eingabe quadriert und überprüft, ob sie kleiner als $\frac{1}{2}$ ist. (d) Für ein gegebenes w kann man M_w auf 1, 10 und 111 gleichzeitig laufen lassen – wenn eine dieser Eingaben terminiert, akzeptieren wir. (e) Die Ungleichung $(7x-31)^2 \leq 8$ hat keine ganzzahlige Lösung, da das Polynom sein Minimum bei $4 < \frac{31}{7} < 5$ annimmt, aber $(7 \cdot 4 - 31)^2 = 9$ und $(7 \cdot 5 - 31)^2 = 16$ beide größer als 8 sind. Die Eigenschaft ist also immer wahr. (f) Wir können über alle Wörter v iterieren und M_w jeweils für $|v|$ Schritte simulieren, die Sprache ist also semi-entscheidbar. Aber nicht entscheidbar (man kann von $\mathcal{H}_0$ reduzieren, indem man die Eingabe ignoriert).

Aufgabe H10.2. (*Rote Zahlen schreiben*) 3+2 Punkte

Heute ist „Tag der offenen Tür“ in Theos Schule und Dora besucht ihn. Es geht um Farbenzahlen, und besonders um die Farbe Rot. Theo hat leider letzte Woche nicht so gut aufgepasst, da er damit beschäftigt war, die Sandkörner in seinem Sandkasten zu zählen. Er kann sich nur noch daran erinnern, dass eine Zahl genau dann rot ist, wenn ihr größter echter Teiler rot ist. Was aber mit Zahlen passiert, die keine echten Teiler haben, weiß er nicht. Dora möchte ihm helfen, hat aber in der Kindergartenvorlesung bisher nur gelernt, welche *Prim*zahlen rot sind. Können Dora und Theo zusammen die gestellten Aufgaben lösen?

Sei $R \subseteq \mathbb{N}_{\geq 2}$ die Menge der roten Zahlen, und P die Menge der Primzahlen. Zur Erinnerung: Für $x, y \in \mathbb{N}_{\geq 1}$ sagen wir „x ist ein *Teiler* von y“, oder $x \mid y$, wenn es ein $k \in \mathbb{N}$ mit $xk = y$ gibt. Wenn zusätzlich $1 < x < y$, dann ist x ein *echter Teiler* von y. Falls x der größte echte Teiler von y ist, gilt also $y \in R \Leftrightarrow x \in R$.

(a) Theo möchte sich ein Verfahren ausdenken, mit dem er die Aufgaben lösen kann. Schreiben Sie also ein `WHILE`-Programm, das die charakteristische Funktion χ_R berechnet (siehe Definition 5.28). Verwenden Sie den zusätzlichen Befehl dora; nach Ausführung von dora wird x_0 auf 1 gesetzt, falls $x_1 \in R \cap P$, sonst auf 0. Erklären Sie ihren Ansatz in natürlicher Sprache.

Sie dürfen alle Makros aus der Vorlesung verwenden, sowie eigene definieren. Für eine beliebige *entscheidbare* Menge M und Variablennamen x, y dürfen Sie zudem $x := y \in M$ als Makro schreiben, das x auf $\chi_M(y)$ setzt. Sie dürfen auch $x := y \notin M$ verwenden, mit analoger Bedeutung.

(b) Zeigen Sie, dass R reduzierbar auf $R \cap P$ ist, indem sie eine berechenbare, totale Funktion $f : \mathbb{N} \to \mathbb{N}$ angeben, sodass $x \in R \Leftrightarrow f(x) \in R \cap P$ für jedes $x \in \mathbb{N}$ gilt.

Lösungsskizze. **(a)** Jede Zahl $x_1 \in \mathbb{N}_{\geq 2}$ ist entweder eine Primzahl, oder besitzt einen echten Teiler. Solange x_1 also keine Primzahl ist, ersetzten wir es durch seinen größten Teiler. Dies beeinflusst nicht, ob $x_1 \in R$. Danach können wir einfach dora aufrufen.

Um uns das Leben leichter zu machen, definieren wir das Makro **while** $x > n$ **do** S **end** für Variablennamen x, $n \in \mathbb{N}$ und Programm S als

$s := x - n$
while $s \neq 0$ **do**
  S
  $s := x - n$
end

Hier ist s eine temporäre Variable, die nirgendwo sonst verwendet wird. Unser Programm ist nun

```
s := x_1 ∉ P ∪ {0, 1}
while s ≠ 0 do
    z := x_1 − 1
    while z > 1 do
        t := x_1 MOD z
        if t = 0 then
            x_1 := z
            z := 0
        end
        z := z − 1
    end
    s := x_1 ∉ P
end
dora
```

(b) Eine einfache Lösung ist, f als die Funktion zu definieren, die von dem `WHILE`-Programm aus (a) ausgerechnet wird, wenn man die letzte Zeile (mit dem dora Befehl), ersetzt durch $x_0 := x_1$. Dann ist f offensichtlich berechenbar. Außerdem ist f total, da das Programm immer terminiert (die innere Schleife kann für höchstens z Iterationen laufen, die äußere höchstens für x_1). Da das ursprüngliche Programm (mit dem dora-Befehl) die Funktion χ_R berechnet, muss das modifizierte Programm eine Funktion f berechnen, die $x \in R \Leftrightarrow f(x) \in R \cap P$ erfüllt.

Als alternative Lösung können wir uns überlegen, was genau das wiederholte Berechnen des größten echten Teilers bewirkt. Sei $x \in \mathbb{N}_{\geq 2}$ beliebig, und y, z jeweils der kleinste und der größte echte Teiler von x. Wir behauptet nun $yz = x$. Dies ist äquivalent zu $z = x/y$. Da y ein Teiler von x ist, gilt $x/y \in \mathbb{N}$, und da $(x/y) \cdot y = x$ ist x/y auch ein Teiler von x. Falls es einen kleineren Teiler gäbe, also $z < x/y$, dann wäre x/z auch ein Teiler von x, und $x/z > x/(x/y) = y$. Aber y ist der größte Teiler, dies ist also ein Widerspruch und $x/y = z$ (und somit unsere Behauptung) gilt.

Nun ist z notwendigerweise eine Primzahl (sonst gäbe es einen kleineren echten Teiler), und zwar die kleinste Primzahl, die x teilt. Wenn wir x als Produkt von Primzahlen schreiben (nach Größe sortiert), also $x = p_1 \cdot_2 \cdot \ldots \cdot p_k$ für $p_1, \ldots, p_k \in P$ mit $p_1 \leq p_2 \leq \ldots \leq p_k$, dann ist $z = p_1$ und $y = p_2 \cdot \ldots \cdot p_k$. Nun lässt sich leicht sehen, dass die wiederholte Anwendung dieses Prozesses p_k als Ergebnis hat, also die größte Primzahl, die x teilt.

Wir definieren unsere Reduktionsfunktion $f : \mathbb{N} \to \mathbb{N}$ nun wie folgt:

$$f(x) := \begin{cases} x & \text{für } x \in P \cup \{0, 1\} \\ \max\{y \in P : y \mid x\} & \text{sonst} \end{cases}$$

Aufgabe H10.3. (*Semi-Entscheidbarkeit*) 2 Punkte

Sei $\Sigma := \{0, 1\}$. Zeigen Sie, dass die Sprache L, definiert als

$$\{w \in \Sigma^* : M_w \text{ hält für zwei unterschiedliche Eingaben in gleich vielen Schritten}\},$$

semi-entscheidbar ist. Sie dürfen ohne Beweis verwenden, dass es berechenbare, surjektive Funktionen $g : \mathbb{N} \to \Sigma^*$ und $h : \mathbb{N} \to \mathbb{N}^k$ gibt, für beliebiges $k \in \mathbb{N}$.

Hinweise: Es ist nicht notwendig (und oft unerwünscht), eine TM formal präzise anzugeben. Achten Sie jedoch darauf, Ihre *Idee* genau zu schildern, sodass einem Leser klar ist, wie eine entsprechende TM auszusehen hätte.

Lösungsskizze. Sei $w \in \Sigma^*$ beliebig. Wir schreiben $M_w[v]\downarrow n$ für M_w hält für Eingabe v in n Schritten. Nun können $w \in L$ äquivalent schreiben als

$$\exists (u, v, n) \in \Sigma^* \times \Sigma^* \times \mathbb{N} : \quad u \neq v \text{ und } M_w[u]\downarrow n \text{ und } M_w[v]\downarrow n$$

Gegeben u, v, n lassen sich $M_w[u]\downarrow n$ und $M_w[v]\downarrow n$ leicht überprüfen, indem man M_w auf der entsprechenden Eingabe für n Schritte simuliert. Unser Programm ist also

for all $(u, v, n) \in \Sigma^* \times \Sigma^* \times \mathbb{N}$
  if $u \neq v \wedge M_w[u]\downarrow n \wedge M_w[v]\downarrow n$ **return**

Es verbleibt noch, zu argumentieren, dass wir über alle Elemente $\Sigma^* \times \Sigma^* \times \mathbb{N}$ iterieren können. Es genügt, eine surjektive, berechenbare Funktion $f : \mathbb{N} \to \Sigma^* \times \Sigma^* \times \mathbb{N}$ zu finden: diese konstruieren wir als $f(n) := (g(h_1(n)), g(h_2(n)), h_3(n))$, wobei g, h wie in der Aufgabenstellung definiert sind, mit $k := 3$.

Aufgabe H10.4. (*Alle Register ziehen*) 0.5+0.5+1 Punkte

Als es gerade den Oeehthto-Gürtel umrundet, wird Tyx'Hxli E'Oozrts Raumschiff von Piraten – geführt vom berüchtigten Dokh'Toa Ivl'Zpaasa – angegriffen. Tyx kann zwar gerade noch entfliehen, steht aber nun vor einem neuen Problem: Sein Bordcomputer wurde schwer beschädigt und fast alle Speicherbänke wurden zerstört. Ohne dessen Rechenkapazität kann Tyx aber unmöglich die quantenstatischen absolutitätstheoretischen Berechnungen ausführen, die für einen Hyperraumsprung notwendig sind. Können Sie Tyx helfen?

Unser Ziel ist es, mit einem `GOTO`-Programm, das nur auf die Variablen x_0 und x_1 zugreifen darf, ein beliebiges Programm zu simulieren. Wir definieren also die *Tyx-`GOTO`*-Programme als die `GOTO`-Programme, die nur die Variablen x_0 und x_1 verwenden.

(a) Beschreiben Sie, wie Sie Zahlen $y_1, ..., y_k \in \mathbb{N}$ über eine einzelne Zahl x darstellen können. Geben Sie also eine berechenbare, injektive Funktion $f : \mathbb{N}^k \to \mathbb{N}$ an, und beschreiben sie, wie sich $f(y_1, ..., y_k)$ verändert, wenn sie einzelne Komponenten von $(y_1, ..., y_k)$ inkrementieren oder dekrementieren.

(b) Implementieren Sie nun ein Tyx-`GOTO` Makro für $y_i := y_i + n$, mit $i \in \{1, ..., k\}$ und $n \in \mathbb{Z}$. Wenn vorher $x_0 = 0$ und $x_1 = f(y_1, ..., y_k)$ gilt (für $y_1, ..., y_k \in \mathbb{N}$), soll nach der Ausführung $x_0 = 0$ und $x_1 = f(y_1, ..., y_{i-1}, \max\{y_i + n, 0\}, y_{i+1}, ..., y_k)$ gelten.

(c) Zeigen Sie, dass das allgemeine Halteproblem für Tyx-`GOTO` Programme unentscheidbar ist, also die Sprache

$$L := \{P\#i : P, i \in \{0,1\}^* \wedge P \text{ ist Tyx-`GOTO` Programm und hält auf } (i)_2\}$$

Lösungsskizze. **(a)** Seien $p_1, ..., p_k$ die ersten k Primzahlen. Dann definieren wir f als

$$f(y_1, ..., y_k) := p_1^{y_1} \cdot p_2^{y_2} \cdot ... \cdot p_k^{y_k}$$

Es ist klar, dass f berechenbar ist. Da die Primfaktorzerlegung einer Zahl eindeutig ist, ist f injektiv. Wenn wir y_i inkrementieren (bzw. dekrementieren) müssen wir f mit p_i multiplizieren (bzw. durch p_i teilen).

(b) Vorab implementieren wir ein Makro für $x_i := 0$:

```
loop: if x_i = 0 goto end
      x_i := x_i − 1
      goto loop
end:  x_0 := x_0
```

Wir implementieren das Makro für die Spezialfälle $n = \pm 1$, die übrigen Fälle erhält man, indem man wiederholt inkrementiert bzw. dekrementiert. Für $n = 1$ müssen wir x_1 einfach mit der Konstanten p_i multiplizieren.

```
loop: x_1 := x_1 − 1
      if x_1 = 0 goto end
      x_0 := x_0 + p_i
      goto loop
end:  x_1 := x_0
      x_0 := 0
```

Für $n = -1$ ist das Vorgehen etwas komplizierter. Hier müssen wir überprüfen, ob x_1 durch p_i teilbar ist. Falls dies nicht der Fall ist, gehen wir zu fail und machen so die vorherigen Veränderungen rückgängig.

```
loop: if x_1 = 1 goto fail
        ⋮
      if x_1 = p_i − 1 goto fail
      x_1 := x_1 − p_i
      if x_1 = 0 goto end
      x_0 := x_0 + 1
      goto loop
end:  x_1 := x_0
      x_0 := 0
      goto done
fail: x_1 := x_1 + p_i
      x_0 := x_0 − 1
      if x_0 = 0 goto done
      goto fail
done: x_0 := x_0
```

(c) Wir wollen zunächst ein Makro $\mathsf{iszero}(y_i)$ implementieren. Wenn vorher $x_0 = 0$ und $x_1 = f(y_1, ..., y_k)$ gilt (für $y_1, ..., y_k \in \mathbb{N}$), soll nach der Ausführung x_0 auf 1 gesetzt werden, falls $y_i = 0$, sonst auf 0. Dafür müssen wir nur unser Programm für $n = -1$ aus (b) leicht anpassen:

```
loop: if x_1 = 1 goto fail
        ⋮
      if x_1 = p_i − 1 goto fail
      x_1 := x_1 − p_i
```

```
        if x_1 = 0 goto end
        x_0 := x_0 + 1
        goto loop
end:    x_1 := x_0
        x_0 := 0
        y_i := y_i + 1
        goto done
fail:   x_1 := x_1 + p_i
        x_0 := x_0 − 1
        if x_0 = 0 goto zero
        goto fail
zero:   x_0 := 1
done:   x_0 := x_0
```

Abhängig davon, ob wir in fail oder done landen, setzten wir x_0. (Wir verwenden $x_0 := 1$ als Makro für $x_0 := 0;\ x_0 := x_0 + 1$.) Im letzteren Fall müssen wir außerdem die Veränderung an x_1 rückgängig machen. Nun können wir ein Makro für „**if** $y_i = 0$ **goto** M“ implementieren:

```
        iszero(y_i)
        if x_0 = 0 goto skip
        x_0 := 0
        goto M
skip:   x_0 := x_0
```

Um beliebige GOTO-Programme simulieren zu können, implementieren wir $y_i := y_j$ für $j \neq i$ als Makro. Hierfür ist y_k eine temporäre Variable, die sonst nicht verwendet wird.

```
drain:  if y_j = 0 goto flood
        y_k := y_k + 1
        y_j := y_j − 1
        goto drain
flood:  if y_k = 0 goto done
        y_k := y_k − 1
        y_i := y_i + 1
        y_j := y_j + 1
        goto flood
done:   x_0 := x_0
```

Schließlich benötigen wir noch ein Makro für „**if** $y_i = n$ **goto** M“:

```
        y_k := y_i − (n − 1)
        if y_k = 0 goto skip
        y_k := y_k − 1
        if y_k = 0 goto M
skip:   x_0 := x_0
```

Nun können wir beliebige GOTO-Programme simulieren. Wir zeigen, dass L unentscheidbar ist, indem wir vom Halteproblem auf leerem Band, $\mathcal{H}_0$, reduzieren. Sei $w \in$

$\{0,1\}^*$ also beliebig. Mir der Prozedur aus der Vorlesung konvertieren wir M_w in ein GOTO-Programm P. Insbesondere terminiert P auf Eingabe 0 genau dann, wenn M_w auf leerem Band hält. Wir ersetzen nun jede Variable x_i in P durch y_i und erhalten P'. Dies konvertieren wir nun mit obigen Makros zu einem Tyx-`GOTO` Programm P''. Wenn wir P'' mit Eingabe $x_1 := 1$ laufen lassen, simuliert es P' auf Eingabe $y_1 = y_2 = ... = y_k = 0$, und terminiert also genau dann, wenn M_w auf leerem Band hält.

Somit haben wir $\mathcal{H}_0$ auf L reduziert. Da $\mathcal{H}_0$ unentscheidbar ist, ist es auch L.