Prof. Javier Esparza
Philipp Czerner, Martin Helfrich

Technische Universität München
Lehrstuhl für Theoretische Informatik

Einführung in die Theoretische Informatik

Sommersemester 2021 – Hausaufgabenblatt 9

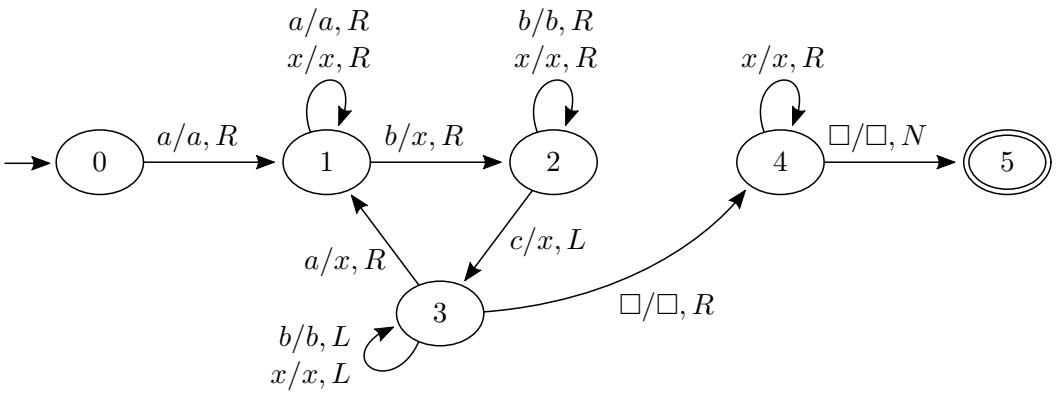
- Sei $\Phi := \{\{1\}, \{2, 3\}, \{4\}\}$. Nach dem Abgabedatum werden wir für jede Menge $A \in \Phi$ eine zufällige Aufgabe $a \in A$ wählen und korrigieren.

  Update: Wir korrigieren H9.1, H9.2 und H9.4
- 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.
- $0 \in \mathbb{N}$

Aufgabe H9.1. (*Jeu de mots*) 1+1+1 Punkte

Ihr Freund, der Archäologe Jasper Vazarie, hat wieder einige Programme ausgegraben, und braucht Ihre Hilfe, um deren Bedeutung zu verstehen.

(a) Folgende Maschine wurde als Grabbeigabe (neben zwei Ringen) in Uruk gefunden. Welche Sprache $L \subseteq \{a, b, c\}^*$ akzeptiert sie?



(b) Überlieferungen zufolge hatte der Hochpriester von Nakbé folgendes Programm verwendet, um seine Entscheidungen zu begründen. Welche Funktion $f : \mathbb{N} \to \mathbb{N}$ berechnet es?

```
z := x_1 + 6
while x_1 ≠ 0 do
    w := z + 0
    while w ≠ 0 do
        y := y + 1
        w := w − 1
    end
    x_1 := x_1 − 1
end
x_0 := y − 91
```

(c) Folgende Wegbeschreibung wurde neben einer römischen Straße im heutigen Italien entdeckt. Welche Funktion $f : \mathbb{N} \to \mathbb{N}$ berechnet sie?

```
Geata:  if x_1 = 0 goto Rom
        y := x_1
        z := 1
        k := y MOD 2
Labici: if k = 1 goto Ardea
        y := y DIV 2
        z := z * 2
        k := y MOD 2
        goto Labici
Ardea:  x_1 := x_1 − z
        x_1 := x_1 * 2
        x_0 := x_0 + 1
        goto Geata
Rom:    HALT
```

Beachten Sie, dass hierbei die syntaktischen Abkürzungen von Folie 245 verwendet wurden. Insbesondere ist $x \,\texttt{DIV}\, y$ die (abgerundete) Division von x durch y, und $x \,\texttt{MOD}\, y$ der verbleibende Rest.

Lösungsskizze.

(a) Die TM sollte die Sprache $\{a^n b^{n+1} c^{n+1} : n \in \mathbb{N}_{>0}\}$ akzeptieren, tut es aber leider nicht.

(b) Zuerst vereinfachen wir das Programm zu

```
z := x_1 + 6
while x_1 ≠ 0 do
    y := y + z
    x_1 := x_1 − 1
end
x_0 := y − 91
```

und dann zu

```
z := x_1 + 6
y := z x_1
x_0 := y − 91
```

Nun ist klar, dass das Programm die Funktion $f(x) := \max\{(x+6)x - 91, 0\}$ berechnet. Äquivalent lässt sich dies auch folgendermaßen schreiben:

$$f(x) := \begin{cases} x^2 + 6x - 91 & \text{für } x \geq 7 \\ 0 & \text{sonst} \end{cases}$$

(c) Nach Labici steht in z die maximale Zahl der Form 2^i, sodass $x_1 2^{-i}$ ganzzahlig ist. Insbesondere ist $x_1 2^{-i}$ damit ungerade, und in der Binärdarstellung von $x_1 - z$ ist ein Bit weniger gesetzt als in x_1. Die Zeile $x_1 := x_1 * 2$ ändert daran nichts, und wir erreichen irgendwann den Fall, in dem kein Bit von x_1 gesetzt ist, ergo $x_1 = 0$. Es führen also alle Wege nach Rom, und x_0 enthält die Anzahl der 1-Bits von x_1. Die Funktion f ist also $f(x) := |\, bin(x) \,|_1$.

Aufgabe H9.2. (*Gelangwhiled?*) 1+1+1+1+2 Punkte

Doras Kindergartenprofessor steht leider im Stau, und die Kindergartenkinder wissen nicht, was sie mit ihrer Zeit anfangen sollen. Ehe es einen Aufstand gibt, beschließen Sie die Kinder zu unterhalten, indem Sie die Konvertierung von `WHILE`-Programmen zu deterministischen Turingmaschinen erklären.

(a) Sei $\Sigma := \{0, 1\}$. Geben Sie eine TM M^0 an, die die Funktion $f : \Sigma^* \to \Sigma^*$ berechnet, mit $f(w) = 0$ für alle w.

(b) Geben Sie eine TM M^- an, die eine Funktion $f : \Sigma^* \to \Sigma^*$ berechnet, wobei $f(bin(i)) := bin(i-1)$ für alle $i \in \mathbb{N}_{>0}$ gilt, und $f(0) := 0$.

(Wie aus der Vorlesung bekannt bezeichnet $bin(i)$ die Binärdarstellung von i, also das kürzeste $w \in \{0, 1\}^+$ mit $(w)_2 = i$.)

Im Folgenden seien die TMs aus (a) und (b) so modifiziert, dass sie auf einem beliebigen (aber festen) Band einer k-Band-TM operieren. Wir nennen die Version von M^-, die auf Band i operiert, $\mathsf{dec}(i)$. Ebenso sei $\mathsf{setzero}(i)$ die Version von M^0, die auf Band i arbeitet. Zusätzlich verwenden wir $\mathsf{inc}(i)$ für die angepasste TM aus Beispiel 5.13 (Binär +1), und $\mathsf{iszero}(i)$ für die „Band $i = 0$?“ TM von Folie 241.

(c) Seien $n, i \in \mathbb{N}$ fix. Konstruieren Sie nun eine k-Band TM $\mathsf{add}(i, n)$, die die Binärzahl auf Band i um n erhöht, und eine k-Band TM $\mathsf{sub}(i, n)$, die die Binärzahl auf Band i um n senkt (statt negativen Ergebnissen ergibt sich 0).

Hinweis: Hier (und im Folgenden) bietet sich die Notation zur sequentiellen Komposition aus der Vorlesung an (Folie 240).

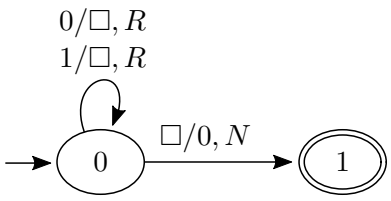
(d) Konstruieren Sie eine k-Band TM $\mathsf{copy}(i, j)$, die das `WHILE`-Programm $x_i := x_j$ simuliert.

(e) Konvertieren Sie nun folgendes `WHILE`-Programm, das eine Funktion $f : \mathbb{N} \to \mathbb{N}$ berechnet, zu einer k-Band TM:

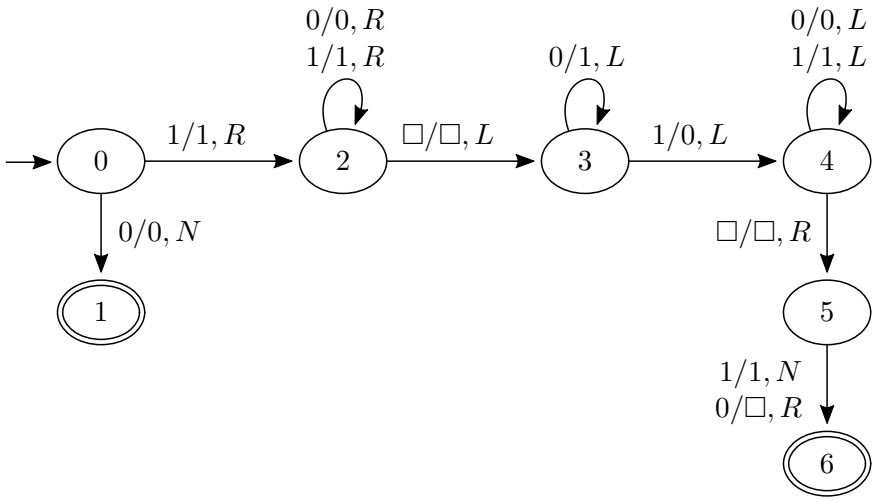
```
while x_1 ≠ 0 do
    x_2 := 0
    while x_1 ≠ 0 do
        x_1 := x_1 − 2
        x_2 := x_2 + 1
    end
    x_1 := x_2
    x_0 := x_0 + 1
end
```

Lösungsskizze.

(a)



(b)



(c) Wir schalten einfach $\mathsf{inc}(i)$ (bzw. $\mathsf{dec}(i)$) n-mal hintereinander:

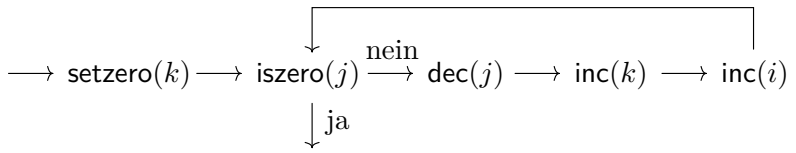
$$\mathsf{add}(i,n)\colon \qquad \to \mathsf{inc}(i) \to \mathsf{inc}(i) \to \cdots \to \mathsf{inc}(i) \to \quad (n\text{-mal})$$

$$\mathsf{sub}(i,n)\colon \qquad \to \mathsf{dec}(i) \to \mathsf{dec}(i) \to \cdots \to \mathsf{dec}(i) \to \quad (n\text{-mal})$$

Alternativ kann man die TMs auch rekursiv definieren, z.B. $\mathsf{add}(i,1) := \mathsf{inc}(i)$ und

$$\mathsf{add}(i,n)\colon \qquad \to \mathsf{add}(i,n-1) \to \mathsf{inc}(i) \to$$

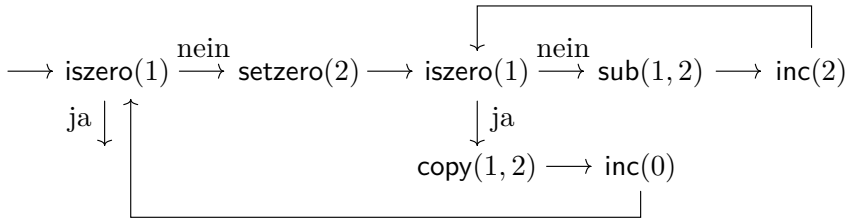
(d) Wir wählen uns eine Hilfsvariable x_k. Zuerst implementieren wir eine TM $\mathsf{move}(k,i,j)$, die das Programm $x_k := x_j;\ x_i := x_i + x_j;\ x_j := 0$ implementiert:



Die TM für $x_i := x_j$ erhalten wir nun als

$$\longrightarrow \mathsf{move}(k,i,j) \longrightarrow \mathsf{move}(i,j,k) \longrightarrow$$

(e)



Aufgabe H9.3. (*Gehe zu `GOTO`, gehe nicht über `WHILE`...*) 1+1+1+1+2 Punkte

In dieser Aufgabe wollen wir uns die Konvertierung von Turingmaschinen zu `GOTO`-Programmen genauer ansehen und diese auf direktem Wege implementieren. Anders als in der Vorlesung benutzen wir also nicht, dass wir `WHILE` über `GOTO` simulieren können. Beachten Sie insbesondere, dass sie in der folgenden Aufgabenteilen nur Makros verwenden dürfen, die wir ohne Simulation von `WHILE`-Programmen darstellen können.

(a) Implementieren Sie ein GOTO-Programm $\mathsf{addmul}(x, y, b)$. Die Parameter x, y sind Namen von Variablen, und $b \in \mathbb{N}_{\geq 2}$ ist eine Zahl. Das Programm soll y mit b multiplizieren, und das Ergebnis auf x addieren.

(b) Implementieren Sie ein GOTO-Programm $\mathsf{divmod}(x, y, z, b)$. Die Parameter x, y, z sind Namen von Variablen, und $b \in \mathbb{N}_{\geq 2}$ ist eine Zahl. Das Programm soll z durch b teilen, das (abgerundete) Ergebnis in x schreiben und den Rest in y. Es soll also $b \cdot x' + y' = z$ und $y' < b$ gelten, wobei x', y' den Wert der Variablen nach der Ausführung bezeichnen.

Fortan können wir $x := y * b + z$ als Makro für $x := z$; $\mathsf{addmul}(x, y, b)$ schreiben (mit x, y, z Variablen, und $b \in \mathbb{N}_{\geq 2}$).

Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ eine TM, mit $\Gamma = \{\square\} \cup \{a_1, ..., a_{b-1}\}$ für ein $b \in \mathbb{N}_{\geq 2}$. Um die spätere Notation zu vereinfachen setzen wir $a_0 := \square$. Wie in der Vorlesung wollen wir einen Bandinhalt $w \in \Gamma^*$ als Zahl darstellen, wir definieren also $f : \Gamma^* \to \mathbb{N}$ über

$$f(\varepsilon) := 0 \qquad \text{und} \qquad f(a_k w) := k + b \cdot f(w) \;\text{ für } k \in \{0, ..., b-1\}, w \in \Gamma^*$$

(c) Ist f surjektiv? Injektiv? Begründen Sie Ihre Antworten kurz.

(d) Implementieren Sie $\mathsf{switch}(x, c_1, M_1, c_2, M_2, ..., c_k, M_k)$, wobei x eine Variable ist, $c_1, ..., c_k \in \Gamma$ paarweise verschieden, und $M_1, ..., M_k$ Marken. Falls $x = f(c_i w)$ für ein $i \in \{1, ..., k\}$ und $w \in \Gamma^*$, soll das Programm zu M_i springen. Ansonsten soll es halten. switch schaut sich also das erste Zeichen des Wortes, das in x gespeichert wird, an, und springt dann zu der entsprechenden Markierung.

(e) Konstruieren Sie schließlich für jedes $q \in Q$ ein Programm der Form $M_q \colon P_q$. Wie in der Vorlesung verwenden wir x und y, um den Bandinhalt zu speichern. In Konfiguration $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$ würde also $x = f(\alpha^R)$ und $y = f(\beta)$ gelten. Sei $C' = (\alpha', r, \beta')$ die eindeutige Konfiguration mit $(\alpha, q, \beta) \to_M (\alpha', r, \beta')$. Nach Ausführung von P_q soll $x = f((\alpha')^R)$ und $y = f(\beta')$ gelten, und Programm P_r ausgeführt werden. Falls ein solches C' nicht existiert, soll das Programm stattdessen anhalten. Intuitiv soll P_q also einen Schritt von M aus Zustand q simulieren.

Sei $\{q_0, ..., q_n\} := Q$. Mit folgendem Programm können wir nun M simulieren:

$$M_{q_0} \colon P_{q_0}; \qquad M_{q_1} \colon P_{q_1}; \quad \cdots \quad M_{q_n} \colon P_{q_n}$$

Hinweis: Sie können $\delta_1, \delta_2, \delta_3$ verwenden, um auf die einzelnen Komponenten von δ zuzugreifen, es gilt also $\delta(q, a) = (\delta_1(q, a), \delta_2(q, a), \delta_3(q, a))$ für $q \in Q$, $a \in \Gamma$.

Lösungsskizze.

(a) Um Seiteneffekte zu vermeiden, kopieren wir y erst in eine neue Variable y'.

```
        y' := y
loop:   if y' = 0 goto end
        x := x + b
        y' := y' − 1
        goto loop
end:    HALT
```

(b) Wir implementieren $\mathsf{divmod}(x, y, z, b)$ über wiederholte Subtraktion.

$x := 0$
$y := z$
loop: $w := y - (b-1)$
if $w = 0$ **goto** end
$x := x + 1$
$y := y - b$
goto loop
end: `HALT`

(c) Die Funktion f ist surjektiv, da jede natürliche Zahl bezüglich jeder Basis $b \in \mathbb{N}_{\geq 2}$ dargestellt werden kann. f ist jedoch nicht injektiv, da $f(\varepsilon) = 0 = f(\square)$. Allgemein lässt sich $\square$ vorne hinzufügen, ohne f zu ändern.

(d) Das erste Zeichen ist einfach der Rest bei Division durch b. Wir schreiben $\gamma(a_i) := i$ für $a_i \in \Gamma$.

$\mathsf{divmod}(w_1, w_2, x, b)$
if $w_2 = \gamma(c_1)$ **goto** M_1
if $w_2 = \gamma(c_2)$ **goto** M_2
$\vdots$
if $w_2 = \gamma(c_k)$ **goto** M_k
`HALT`

(e) Zuerst implementieren wir ein Programm $\mathsf{delta}_q(x, y, M_L, M_R, M_N)$, das x auf das $i \in \{0, ..., b-1\}$ setzt, sodass $\delta_2(q, a_y) = a_i$ gilt, und danach zu $M_{\delta_3(q,a_y)}$ springt. Falls $\delta(q, a_y)$ undefiniert ist, hält das Programm stattdessen. Seien dazu $c_1, ..., c_k \in \Gamma$ die Zeichen, sodass $\delta(q, c_i)$ definiert ist, für $i \in \{1, ..., k\}$.

$\mathsf{switch}(y, c_1, M_1, ..., c_k, M_k)$
M_{q1}: $x := \delta_2(q, a_1)$; **goto** $M_{\delta_3(q,a_1)}$
M_{q2}: $x := \delta_2(q, a_2)$; **goto** $M_{\delta_3(q,a_2)}$
$\vdots$
M_{qk}: $x := \delta_2(q, a_k)$; **goto** $M_{\delta_3(q,a_k)}$

Das eigentliche Programm ist nun wie folgt.

M_q: $\mathsf{divmod}(y, c, y, b)$
$\mathsf{delta}_q(c', c, M_{qL}, M_{qR}, M_{qN})$
M_{qL}: $y := y * b + c'$
$\mathsf{divmod}(x, c'', x, b)$
$y := y * b + c''$
goto $M_{q'}$
M_{qR}: $x := x * b + c'$
goto $M_{q'}$
M_{qN}: $y := y * b + c'$
$M_{q'}$: $\mathsf{switch}(c, c_1, M_{\delta_1(q,c_1)}, ..., c_k, M_{\delta_1(q,c_k)})$

Aufgabe H9.4. (*Permutation*) 1+2 Punkte

Sei $\Sigma := \{a, b\}$. Für ein Wort $w \in \Sigma^*$ mit $w = w_1 w_2 ... w_n$ definieren wir nun die Menge der *Permutationen* von w als

$$\mathrm{Perm}(w) := \{w_{f(1)} w_{f(2)} ... w_{f(n)} \mid f : \{1, ..., n\} \to \{1, ..., n\} \text{ bijektiv}\}$$

Beispielweise gilt $\text{Perm}(abaa) = \{aaab, aaba, abaa, baaa\}$. Wir erweitern dies auf Sprachen, für $L \subseteq \Sigma^*$ gilt also $\text{Perm}(L) := \bigcup_{w \in L} \text{Perm}(w)$.

(a) Zeigen oder widerlegen Sie: Für **jede** Sprache L gibt es eine TM M, die $\text{Perm}(L)$ akzeptiert.

(b) Zeigen Sie, dass $\text{Perm}(L)$ kontextfrei ist, wenn L regulär ist.

Lösungsskizze. **(a)** Die Aussage ist falsch. Wir zeigen, dass es überabzählbar viele Sprachen $\text{Perm}(L)$ gibt. Daraus folgt die Aussage, denn – wie aus der Vorlesung bekannt – es gibt nur abzählbar viele TMs, und jede TM akzeptiert genau eine Sprache.

Zuerst beobachten wir, dass $\text{Perm}(L) = L$ für $L \subseteq \{a\}^*$. Es genügt also zu zeigen, dass es überabzählbar viele unäre Sprachen gibt. Eine Sprache $L \subseteq \{a\}^*$ können wir identifizieren mit einer Funktion $f : \mathbb{N} \to \{0, 1\}$, die für jedes $n \in \mathbb{N}$ angibt, ob $a^n \in L$. Nach Satz 5.8 ist die Menge der Funktionen $f : \mathbb{N} \to \{0, 1\}$ überabzählbar.
(b) Hier gibt es zwei Ansätze: Man kann aus einem DFA M für L einen PDA M' direkt konstruieren. M' hat als Zustände genau die Zustände von M, und verwendet den Stapel als Zähler. Es gibt folgende Transitionen:

- M' kann ein a einlesen, und den gleichen Übergang wie M ausführen.
- M' kann ein b einlesen, und den Zähler um 1 dekrementieren.
- M' kann den b-Übergang von M ausführen, und den Zähler um 1 erhöhen.
- Wenn M' in einem Finalzustand von M ist und der Zähler auf 0 steht, kann M' akzeptieren.

Statt dies formal zu beweisen werden wir aber eine andere Lösung angeben, die Aufgabe H8.5 verwendet.

Sei $f : \Sigma^* \to \mathbb{N}^2$ mit $f(w) := (|w|_a, |w|_b)$. Wir erweitern f auf Sprachen und setzten $f(L) := \{f(w) : w \in L\}$ für $L \subseteq \Sigma^*$. Offensichtlich gilt $w \in \text{Perm}(L) \Leftrightarrow f(w) \in f(L)$, da es nur darauf ankommt, wie viele Zeichen in w vorhanden sind. Für einen regulären Ausdruck s schreiben wir außerdem $F(s) := f(L(s))$.

Sei s nun ein regulärer Ausdruck. Wir zeigen nun, dass $F(s)$ semilinear ist, indem wir mit struktureller Induktion über s beweisen, dass es reguläre Ausdrücke $s_1, ..., s_k$ gibt, sodass $F(s_i)$ linear ist und $s \equiv s_1 \,|\, ... \,|\, s_k$.

- $s = \emptyset$: Hier gilt $F(s) = \emptyset$, wir setzten also $k := 0$.
- $s = \epsilon \vee s = a \vee s = b$: Die Mengen $F(s) = \{(0, 0)\}$, $F(s) = \{(1, 0)\}$ und $F(s) = \{(0, 1)\}$ sind bereits linear.
- $s = t \,|\, u$: Wir wählen $t_1, ..., t_k$ und $u_1, ..., u_l$ entsprechend der Induktionsannahme, es gilt also $s \equiv t_1 \,|\, ... \,|\, t_k \,|\, u_1 \,|\, ... \,|\, u_k$ und somit die Aussage für s.
- $s = tu$: Hier gilt $s \equiv t_1u_1 \,|\, t_1u_2 \,|\, ... \,|\, t_1u_l \,|\, t_2u_1 \,|\, ... \,|\, t_ku_l$. Es genügt also zu zeigen, dass für beliebige i, j die Menge $F(t_iu_j) = \{x + y : x \in F(t_i), y \in F(u_j)\}$ linear ist. Da (nach Induktionsannahme) $F(t_i)$ und $F(u_j)$ linear sind, können wir Vektoren $r, p_1, ..., p_j, r', p'_1, ..., p'_k \in \mathbb{N}^2$ mit

  $$F(t_i) = \{r + \lambda_1 p_1 + ... + \lambda_m p_m : \lambda \in \mathbb{N}^m\}$$
  $$F(u_j) = \{r' + \lambda'_1 p'_1 + ... + \lambda'_n p'_n : \lambda' \in \mathbb{N}^n\}$$

  wählen. (Dies folgt aus der Definition von linear.) Dann gilt

  $$F(t_iu_j) = \{r + r' + \lambda_1 p_1 + ... + \lambda_m p_m + \lambda'_1 p'_1 + ... + \lambda'_n p'_n : \lambda \in \mathbb{N}^m, \lambda' \in \mathbb{N}^n\}$$

Somit ist $F(t_i u_j)$ ebenfalls linear. Insbesondere haben wir auch gezeigt, dass $F(tu)$ linear ist, wenn $F(t)$ und $F(u)$ linear sind. $(*)$

- $s = t^*$: Aus $t \equiv t_1 \,|\, ... \,|\, t_k$ folgt $t^* \equiv (t_1^* ... t_k^*)^*$. Wenn wir also zeigen können, dass $F(t^*)$ linear ist, wenn $F(t)$ linear ist, dann ist $F(t_i^*)$ für jedes i linear, nach $(*)$ ist somit $F(t_1^* ... t_k^*)$ linear, und schließlich auch $F(t^*)$.

  Sei also $F(t)$ linear. Wir wählen wieder $r, p_1, ..., p_j \in \mathbb{N}^2$ mit

  $$F(t) = \{r + \lambda_1 p_1 + ... + \lambda_m p_m : \lambda \in \mathbb{N}^m\}$$

  Da jedes Wort in t^* die Konkatenation von endlich vielen Wörtern aus t ist, folgt

  $$F(t^*) = \{\lambda' r + \lambda_1 p_1 + ... + \lambda_m p_m : \lambda' \in \mathbb{N}, \lambda \in \mathbb{N}^m\}$$

  Also ist $F(t^*)$ linear.

Wir haben gezeigt, dass $F(s)$ für jeden regulären Ausdruck s semilinear ist. Damit ist aus $f(L)$ für jede reguläre Sprache L semilinear, und $\mathrm{Perm}(L)$ ist nach Aufgabe H8.5 kontextfrei.