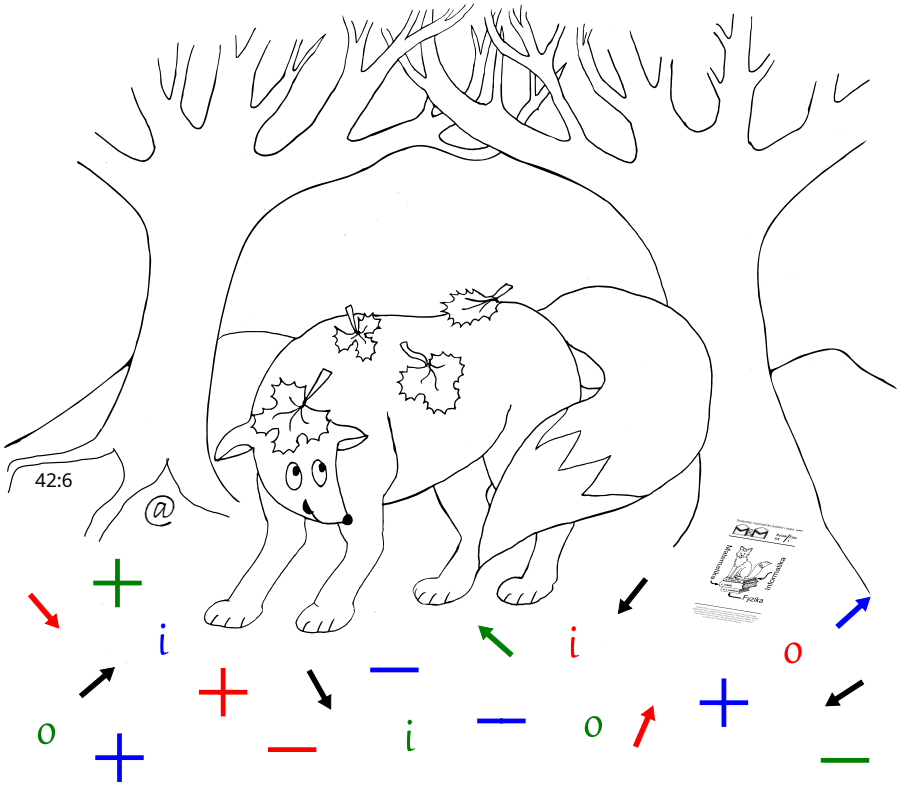
STUDENTSKÝ ČASOPIS A KORESPONDENČNÍ SEMINÁŘ

M&M

Ročník XXX
Číslo 4

MATEMATIKA FYZIKA INFORMATIKA



Uvnitř najdete několik témat a s nimi souvisejících úloh. Zamyslete se nad nimi a pošlete nám svá řešení. My vám je opravíme a ta nejzajímavější z nich otiskneme. Nejlepší řešitele zveme na podzim a na jaře na soustředění.

Milý řešiteli,

neváhej a co nejdříve se pusť do řešení problémů a úloh! První deadline tohoto čísla je zároveň deadlinem pro soustředění, které proběhne od 20. do 28. dubna. Body za řešení, která zašlete do prvního deadlinu, tedy započítáme při výběru účastníků. To si nenech ujít!

A co tě v tomto čísle čeká?

Termodynamika nabízí vzorová řešení úloh ze 3. dílu. Jejich čtení si pak můžeš osladit článkem, ve kterém se Mgr.^{MM} Monika Drexlerová věnuje měření měrné tepelné kapacity čokolády.

V tématku *Lean* si též můžeš prostudovat vzorová řešení z minulého dílu, kromě toho tě ale čeká mnoho nového. Zabrousíme společně k základům logiky, teorie množin a funkcí a to vše budeš moct využít při řešení mnoha zajímavých úloh.

S tématkem *Hex* se v tomto čísle loučíme, avšak nesmutni. Pokud by se ti po něm zastesklo, můžeš se mu stále věnovat, například hledat nové výherní strategie.

Naopak *Genetika* se vrací! Naučíš se kreslit rodokmeny a dozvíš se, k čemu je to dobré. Zkus si nakreslit rodokmen svojí vlastní rodiny!

Na závěr pro tebe máme zcela nové tématko *FlatFox#*. Naprogramuj cestu lišky lesem a vyřeš tím některou z úloh! Narazí po cestě tvoje liška na bludný kořen?

Ať ti řešení jde a na viděnou na soustředění!

Tvoji organizátoři

Obsah

Téma 1 – Termodynamika **3**

Téma 2 – Programování a dokazování v Leanu **10**

Téma 3 – Hex **30**

Téma 4 – Genetika **31**

Téma 5 – FlatFox# **43**

Zadání a řešení témat

1. deadline: 1. března 2024 | 2. deadline: ~~26. března~~ 2. dubna 2024
Řešení odevzdaná do 1. března se započítají pro účast na soustředění.

Téma 1 – Termodynamika

Tento díl neobsahuje nové zadání, autory zavalily školní povinnosti, nicméně další číslo bude obsahovat pokračování, představíme si entropii a trochu zabrousíme i do statistické mechaniky. Stále se však můžete pustit do řešení následujících problémů z předminulého čísla:

Problém 2.4: *Popište detailněji, jak fungují moderní kalorimetry a jejich výhody a limity. Jak vypadá vzorek, který je ideální adept na měření kalorimetrem? Jaké vlastnosti by vzorek mít neměl?*

Problém 2.5: *Zkuste experimentálně změřit hodnotu měrné tepelné kapacity. Může vám na to posloužit jednoduchý model kalorimetru zvaný Blackův kalorimetr. V principu jde o to, že z měření určíte, kolik tepla se vymění mezi ledovou nádobou (to odpovídá množství roztáté vody) a tělesem. Budou se vám lépe měřit látky s vysokou nebo nízkou měrnou tepelnou kapacitou? Zkuste zdůvodnit. Pokud najdete jiný způsob měření c_V, popište jej a vyzkoušejte jej.*

Dále si můžete přečíst článek od Mgr.[MM] Moniky Drexlerové na téma měrné tepelné kapacity čokolády. Můžete se tímto článkem inspirovat a udělat si svůj vlastní experiment. Pořád platí, že hezká řešení otiskneme.

V tomto díle vám představujeme vzorová řešení úloh 3.2 a 3.3 od Doc.[MM] Jany Uglickich spolu s autorským řešením úlohy 3.1.

Vzorová řešení 3. dílu

Úloha 3.1

Zadání:
Jak by vztah pro entalpii vypadal v diferenciální formě pro proměnný tlak systému během přechodu z počátečního stavu do konečného? Odvoďte jej a popište, co znamenají jednotlivé veličiny v odvozeném vztahu.

Řešení:
Definice vnitřní energie je v diferenciálním tvaru $\mathrm{d}U = Q + W$. Znaménko d značí (nekonečně) malou změnu hodnoty mezi dvěma stavy.

Pokud budeme uvažovat jen objemovou práci (zanedbáme jaderné přeměny, chemické reakce v systému atd.), platí: $\mathrm{d}U = Q - p \cdot \mathrm{d}V$.

Definice entalpie zní $H = U + pV$. Tuto si přepíšeme do diferenciálního tvaru: $\mathrm{d}H = \mathrm{d}U + p \cdot \mathrm{d}V + V \cdot \mathrm{d}p$. Dosadíme za $\mathrm{d}U$: $\mathrm{d}H = Q - p \cdot \mathrm{d}V + p \cdot \mathrm{d}V + V \cdot \mathrm{d}p = Q + V \cdot \mathrm{d}p$.

Vztah pro entalpii H je tedy následující: $\mathrm{d}H = Q + V \cdot \mathrm{d}p$. Platí za podmínky, že se nekoná jiná práce než objemová, jde tedy použít k popisu spousty dějů a systémů, které tuto podmínku splňují. Jde například o systémy, ve kterých neprobíhají chemické rekce a molekuly se sebou na molekulární úrovni interagují zanedbatelně (nebo neinteragují vůbec, ale to je možné jen v teorii). Q je teplo přijaté či odevzdané systémem, p a V jsou tlak a objem v systému.

Úloha 3.2

Zadání:

Reakce $C_2H_4O\ (g) + H_2O\ (l) \longrightarrow C_2H_6O_2\ (l)$ probíhá při 25 °C a tlaku 101,325 kPa. Výpočet proveďte pro případ, že všechny látky jsou v plynné (platí stavová rovnice ideálního plynu) a nebo v kapalné fázi tak, jak je uvedeno v reakci. Data při podmínkách reakce:

	C_2H_4O	H_2O	$C_2H_6O_2$
$\Delta H^\circ_{sl}(298{,}15\,\mathrm{K},(g))/(\mathrm{kJ/mol})$	−52,60	−241,827	−389,32
$\Delta H_{výp}(298{,}15\,\mathrm{K})/(\mathrm{kJ/mol})$	26,20	44,0	71,20
$\rho/(\mathrm{g/cm^3})$	0,8610	0,9971	1,1094

1. *Na základě následujících dat vypočtěte standardní změnu entalpie při hydrataci ethylenoxidu na glykol.*

2. *Vypočtěte změnu molárního objemu během reakce.*

3. *Vypočtěte změnu vnitřní energie reakce.*

Pro zisk extra bodů popište, jaké předpoklady platí pro vzorce, co jste použili (zda plyn musí být ideální, zda něco zanedbáváte apod.). Zamyslete se nad tím, kdy vámi vymyšlený vzorec použít nelze.

Řešení od Doc.MM Jany Uglickich:

1. podúloha

Za daných podmínek platí, jak lze přímo vyčíst z tabulky,

$$\Delta H_{\mathrm{C_2H_4O}} = -52{,}6\,\mathrm{kJ \cdot mol^{-1}}.$$

Pro látky v kapalném skupenství platí, že $\Delta H^\circ_{sl}(l) = \Delta H^\circ_{sl}(g) - \Delta H_{výp}$, proto

$$\Delta H_{\mathrm{H_2O}} = -241{,}827 - 44 = -285{,}827\,\mathrm{kJ \cdot mol^{-1}},$$

$$\Delta H_{\mathrm{C_2H_6O_2}} = -389{,}32 - 71{,}2 = -460{,}52\,\mathrm{kJ \cdot mol^{-1}}.$$

Potom $\Delta H = \Delta H_{\mathrm{C_2H_6O_2}} - \Delta H_{\mathrm{C_2H_4O}} - \Delta H_{\mathrm{H_2O}} = -122{,}093\,\mathrm{kJ \cdot mol^{-1}}$.

2. podúloha

O molárním objemu platí

$$V_m = \frac{M}{\rho},$$

kde M je molární hmotnost látky a ρ její hustota. Stačí tedy určit V_1 molární objem reaktantů, V_2 molární objem produktů a následně spočítat

$$\Delta V_m = V_2 - V_1.$$

C_2H_4O budu považovat za ideální plyn, tj. pro něj platí $V_m = V/n = RT/p$.

$$V_1 = \frac{RT}{p} + \frac{M_{H_2O}}{\rho_{H_2O}} = \frac{8{,}314 \cdot 298{,}15}{101325} + \frac{2+16}{0{,}9971} =$$

$$= 0{,}02445\,\mathrm{m^3 \cdot\ mol^{-1}} + 18{,}0524\,\mathrm{cm^3 \cdot\ mol^{-1}} = 0{,}02447\,\mathrm{m^3 \cdot\ mol^{-1}}$$

$$V_2 = \frac{M_{C_2H_6O_2}}{\rho_{C_2H_6O_2}} = \frac{24+6+32}{1{,}1094} = 55{,}89 \cdot 10^{-6}\,\mathrm{m^3 \cdot\ mol^{-1}}$$

$$\Delta V_m = -0{,}0244\,\mathrm{m^3 \cdot\ mol^{-1}}$$

3. podúloha

Změnu vnitřní energie lze vyjádřit jako $\Delta U = \Delta H - p\Delta V$ (pro konstantní tlak, což v tomto případě platí). Dosazením vypočítaných nebo známých údajů do vzorce získáme

$$\Delta U = -122093 - 101325 \cdot (-0{,}0244) = -119620{,}67\,\mathrm{J \cdot\ mol^{-1}} \doteq$$

$$\doteq -119{,}62\,\mathrm{kJ \cdot\ mol^{-1}}.$$

Předpoklady, za kterých vzorce platí

Napadly mě následující okolnosti, které jsou potřeba, aby byl můj postup správný:

- tlak je během celého průběhu reakce konstantní,
- C_2H_4O je ideální plyn.

Úloha 3.3

Zadání:

Při experimentu, jehož cílem bylo určení slučovací entalpie celulózy, bylo zjištěno, že během dokonalého spálení $m = 0{,}6\,g$ polysacharidu celulózy (chemický vzorec $[C_6H_{10}O_5]_{12}$) v kalorimetrické bombě přešlo při $25\,°C$ do okolí teplo $Q = 2000\,J$. Pomocí slučovacích tepel CO_2 (g) a H_2O (l) určete slučovací teplo tohoto polysacharidu. Potřebujete molární hmotnost celulózy M a slučovací tepla při $25\,°C$,

která naleznete v tabulkách nebo na internetu. Reakce probíhá za stálého objemu. Je reakce endotermická nebo exotermická? Proč?

Nápověda: napište si rovnici, objemy látek v kapalném a tuhém stavu zanedbáváme vůči velikosti objemu plynných látek. Člen pV se vyskytuje i v rovnici pro ideální plyn. Při konstantním objemu systému je uvolněné teplo Q během děje rovno změně molární vnitřní energie ΔU°_{sp} podle rovnice: $\Delta U^\circ_{sp} = QM/m$.

Řešení od Doc.MM Jany Uglickich:

Hned v zadání je zmíněno, že teplo bylo **vydáno** – jedná se tedy o exotermickou reakci a vnitřní energie soustavy během ní klesne. ΔU je tedy záporné a teplo Q bude mít záporné znaménko.

Platí

$$\Delta U^\circ_{sp} = \frac{QM}{m},$$

$$\Delta H = H_{\text{produkty}} - H_{\text{reaktanty}} = \Delta U + p\Delta V.$$

Reakci hoření celulózy popisuje rovnice:

$$1\,[\mathrm{C_6H_{10}O_5}]_{12}\,(\mathrm{s}) + 72\,\mathrm{O_2}\,(\mathrm{g}) \longrightarrow 60\,\mathrm{H_2O}\,(\mathrm{l}) + 72\,\mathrm{CO_2}\,(\mathrm{g}).$$

Lze si dohledat slučovací entalpie O_2, H_2O a CO_2:

$$\Delta H_{\mathrm{O_2}} = 0\,\mathrm{kJ}\cdot\mathrm{mol}^{-1}(\text{protože jde o prvek}),$$

$$\Delta H_{\mathrm{H_2O}} = -285{,}827\,\mathrm{kJ}\cdot\mathrm{mol}^{-1},$$

$$\Delta H_{\mathrm{CO_2}} = -393{,}521\,\mathrm{kJ}\cdot\mathrm{mol}^{-1}.$$

Zároveň, pokud zanedbáme objemy kapalných i pevných látek a budeme považovat plynné látky za ideální plyny, potom je objem reaktantů stejný jako objem produktů, takže změna objemu je nulová.

S těmito znalostmi lze sestavit následující rovnici a potom do ní dosadit:

$$\begin{aligned}
\frac{QM}{m} &= 60\Delta H_{\mathrm{H_2O}} + 72\Delta H_{\mathrm{CO_2}} - \Delta H_{[\mathrm{C_6H_{10}O_5}]_{12}} \\
\Delta H_{[\mathrm{C_6H_{10}O_5}]_{12}} &= 60\Delta H_{\mathrm{H_2O}} + 72\Delta H_{\mathrm{CO_2}} - \frac{QM}{m} = \\
&= 60\cdot(-285{,}827) + 72\cdot(-393{,}521) - \frac{(-2)\cdot 1944}{0{,}6} = \\
&= -39003{,}132\,\mathrm{kJ}\cdot\mathrm{mol}^{-1}.
\end{aligned}$$

Slučovací entalpie celulózy $[\mathrm{C_6H_{10}O_5}]_{12}$ je tím pádem $-39003{,}132\,\mathrm{kJ}\cdot\mathrm{mol}^{-1}$.

Pája, Helča a Pavel; `termodynamika@ledoian.cz`
odevzdávejte do odevzdávátka

Měrná tepelná kapacita čokolády 8b

Mgr.[MM] *Monika Drexlerová*

Experiment

Úvod

K měření měrné tepelné kapacity jsem si zvolila čokoládu.

Teorie

Protože čokoláda je amorfní látka, její tání probíhá nerovnoměrně. Její teplota tání je rovněž docela nízká, proto budeme měření provádět okolo teploty 0 °C. Měrnou tepelnou kapacitu budeme měřit pomocí ledu a jeho skupenské přeměny, což jsem počítala jako[1] $334\,\mathrm{kJ\cdot kg^{-1}}$.

Postup a pomůcky

Celkem jsem provedla 4 měření (uvědomuji si, že to není úplně referenční statistický vzorek, ale experiment jako takový byl docela náročný). Vždy jsem zamrazila vodu ve 4 stejně velkých krabicích. Když jsem měla tento led, vložila jsem mezi krabice („hladinou ledu“ k sobě) tabulku čokolády o pokojové teplotě. Druhé dvě krabice jsem dala stejně, jen jsem mezi ně nevložila čokoládu. Tímto jsem měřila množství ledu, které odtálo teplotou v místnosti a ne díky zahřátí se od čokolády (což měříme). Teplotu čokolády jsem v průběhu měřila (infračerveným bezkontaktním teploměrem), dokud nedosáhla teploty 0 °C (zde bylo důležité pohlídat si na začátku i teplotu ledu, protože náš mrazák je chladnější, tedy i led sám se musel chvíli zahřívat). Poté co čokoláda dosáhla této teploty, vyměnila jsem ji za totožnou tabulku pokojové teploty a to jsem provedla 2krát (celkem 3 tabulky, abych měla dostatek čokolády a relativně dobře měřitelný rozdíl hmotností).

Po zahřátí 3 tabulek jsem zvážila obě dvojice krabic (po odlití vody) a zaznamenala jejich rozdíl. Tento proces jsem opakovala 4krát.

Data

Nejistotu ze statistického souboru a měření jsem počítala jako (kde chybu přístroje jsem vzala jako 5 g – raději ať je větší vzhledem k faktorům ovlivňující nepřesnost měření, které rozebírám níže):

$$N_1 = \sqrt{\frac{\sum_{i=1}^{n}(X_j - X_p)^2}{n-1}}$$

$$N_2 = \frac{N_1}{\sqrt{n}}$$

$$N = \sqrt{N_2^2 + N_s^2}$$

[1] Pozn. redakce: Tato číselná hodnota je tabulková hodnota měrného skupenského tepla pro skupenskou přeměnu tání ledu.

Celkovou nejistotu koeficientu jsem poté počítala jako nejistotu v podílu, pro výpočet koeficientu z rovnice (vše jsem převedla na jouly a gramy):

$$mc\Delta T = \Delta m_{\text{ledu}} l_t$$

Kde $m = 100\,\text{g}$ je hmotnost čokolády, c je měrná tepelná kapacita čokolády (vyjde v joulech na gramy na kelviny), ΔT je rozdíl mezi 0, na kterou čokoládu chladíme, a pokojovou teplotou, kterou má čokoláda původně (tedy 20 K), Δm_{ledu} je rozdíl hmotností kontrolní krabice s ledem a testovací krabice s ledem (testovací bude lehčí, protože odtaje více vody – tato voda chladí čokoládu) a $l_t = 334000\,\text{J}\cdot\text{g}^{-1}$ je skupenské teplo tání ledu. Po dosazení hodnot a vyjádření získáme:

$$3 \cdot 100 \cdot c \cdot 20 = \Delta m_{\text{ledu}} \cdot 334000$$

$$c = \frac{\Delta m_{\text{ledu}} \cdot 334000}{3 \cdot 100 \cdot 20}$$

	rozdíl m[g]		
1.	43	průměr [g]	41,75
2.	38	nejistota hmotnosti [g]	5,22
3.	45	nejistota kapacity [J/gK]	0,29
4.	41	c [J/gK]	2,32

Kapacita tedy vyšla $c = (2{,}32 \pm 0{,}29)\,\text{J}\cdot\text{g}^{-1}\cdot\,\text{K}^{-1}$ s tím, že odchylka okolo 12 % vznikla díky velkému rozptylu malého souboru dat a velké chybě přístroje.

Závěr a diskuze

Experiment byl hodně omezen jeho složitostí a náročností na provedení, tomu rovněž odpovídá relativní nepřesnost.

Co se naměřených dat týče, řekla bych, že vyšla docela v pořádku. Jediný zdroj pro měrnou tepelnou kapacitu čokolády, co jsem našla, byl anglicky a uváděl hodnotu[2] $1{,}6\,\text{J}\cdot\text{g}^{-1}\cdot\text{K}^{-1}$, což sice je docela mimo mé měření, nicméně existuje pár vlivů, které tento rozdíl způsobily.

Na krabice, do kterých jsem vkládala čokoládu, jsem podstatně více sahala, tedy i toto teplo mohlo mít efekt. Dále ani samotné měření teploty čokolády nebylo přesné, tedy je možné, že buď teplota ledu nebo tabulky nedosáhla, nebo přesáhla danou hodnotu. Všechny tyto faktory výsledné teplo do rovnice jen přidávají, tedy je pochopitelné, že moje kapacita vyšla vyšší.

K eliminaci ostatních chyb jsem použila identické čokolády, nechala krabice na stejném místě a před každým měřením je nechala chladnout stejnou dobu. Rovněž jsem je vždy dolila na identickou váhu (mezi sebou i s ostatními měřeními).

[2] Zdroj: `https://www.sciencedirect.com/science/article/pii/S1466856421000308`.

Experiment by se dal zlepšit přesnějším vybavením, celkovým odizolováním od okolního prostředí, větším počtem měření, ale zároveň by i vyšel lépe pro látky s větší hodnotou c (nebo látky, co mají větší hustotu – jsou těžší – zvětší faktor hmotnosti, ale i co mají vyšší teplotu tání než čokoláda – mohla bych je zahřát a zvětšit rozdíl teplot). Experiment by šel zkrátka zlepšit jakýmkoliv způsobem, který by udělal rozdíl v odtáté vodě znatelnější od rozdílu, který odtaje jen díky okolní teplotě.

Téma 2 – Programování a dokazování v Leanu

Díl 4: Logika

Strávili jsme tři díly Leanu v říši čísel. V tomhle čísle se zaměříme na základnější matematické věci – co existovalo od pradávna, ještě než bůh stvořil čísla. Proto v tomhle díle najdete mnohem méně sčítání, odčítání, násobení a dělení. Naopak tady najdete spoustu implikací, konjunkcí, kvantifikátorů a množin. Nejspíš shledáte, že když pracujeme s abstraktnějšími věcmi, je Lean více nápomocen (či méně na obtíž; je na vás, jak se na to budete dívat) než třeba při práci s reálnými čísly[3].

Výroková logika

Nejdůležitější logická spojka je implikace. Výraz `P → Q` říká, že když platí `P`, musí platit i `Q`. Abyste napsali šipku ve VS Code, napište `\r` a stiskněte mezerník. Toto je pravdivostní tabulka pro implikaci:

`P`	`Q`	`P → Q`
`True`	`True`	`True`
`True`	`False`	`False`
`False`	`True`	`True`
`False`	`False`	`True`

Některým lidem připadá matoucí, že `False → True` je pravda. Pojďme si to objasnit na příkladu, který demonstruje všechny tři pravdivé situace. Řekli byste, že v kontextu `(x : ℤ)` je tvrzení `(x > 20) → (x > 10)` pravdivé? To nepochybně je! Každé číslo větší než dvacet je určitě větší než deset. Pojďme si tam dosadit nějaké konkrétní hodnoty:

- `(x = 25)` dává `True → True`
- `(x = 15)` dává `False → True`
- `(x = 5)` dává `False → False`

Jedině `True → False` vzniknout nemůže (ani tady ani jinde).

Možná vás zarazilo, proč tu píšeme `True` a `False`, zatímco v prvním díle bylo `true` a `false`. To není chyba. V prvním díle jsme pracovali s typem `Bool`, který se používá pro výpočty. V tomto díle pracujeme s typem `Prop`, který se používá pro tvrzení[4]. Přesný rozdíl mezi typy `Bool` a `Prop` je nad rámec tohoto textu. Můžeme však naznačit, že term typu `Bool` je výraz, který Lean vždycky vyhodnotí na `true` nebo `false`. Typ[5] `Prop` se nehodí používat pro výsledky výpočtů, ale můžeme

[3] Reálně jsou reálná čísla pro reálný počítač hodně složitá.

[4] Anglicky se „tvrzení“ řekne „proposition“.

[5] Ve skutečnosti `Prop` není typ nýbrž univerzum. Ale to tady nechceme rozebírat. Omluvte prosím technické nepřesnosti, kterých se tu dopouštíme.

o něm psát důkazy. Například `Irrational (Real.sqrt 2)` je term typu `Prop`. Dá se o něm dokázat, že se rovná `True`. Nedá se o něm dokázat, že se rovná `False`. V minulém čísle jsme celou dobu pracovali s termy typu `Prop` za hlavní dvojtečkou, aniž bychom se o tom zmiňovali.

Ve zveřejněném kódu[6] si můžete prohlédnout, jak se s implikacemi pracuje jak na úrovni taktik, tak i na úrovni termů[7]. Ve zbytku témátka budeme psát důkazy pomocí taktik, jak jsme to dělali v minulém čísle. Velice užitečná je taktika `apply` pro tzv. zpětné uvažování (od cíle k předpokladům). V okamžiku, kdy `Q` je cíl a máme v lokálním kontextu tvrzení `(pq : P → Q)`, kde `P` a `Q` jsou libovolně složité termy typu `Prop`, můžeme napsáním `apply pq` přeměnit cíl na `P`. Zároveň dochází k dosazení správných konkrétních hodnot,[8] když `P` a `Q` záležely na nějakých společných argumentech.[9] Špatným použitím `apply` se ale můžeme dostat do slepé uličky.

Další logická spojka je konjunkce. Výraz `P ∧ Q` říká, že `P` musí platit a `Q` také musí platit. Ve VS Code napište `\and` a stiskněte mezerník. Toto je pravdivostní tabulka pro konjunkci:

P	Q	P ∧ Q
True	True	True
True	False	False
False	True	False
False	False	False

Pojďme si ukázat jeden z mnoha způsobů jak dokázat, že v konjunkci nezáleží na pořadí:

```
example {P Q : Prop} (predpoklad : P ∧ Q) : Q ∧ P := by
  obtain ⟨p,q⟩ := predpoklad
  constructor
  • exact q
  • exact p
```

První řádek říká, co máme k dispozici a co chceme dokázat. Následující řádek rozebere `(predpoklad : P ∧ Q)` na části. V lokálním kontextu se objeví `(p : P)` a `(q : Q)`. Špičaté závorky, které se používají v `obtain` i některých dalších taktikách, které něco dělají s „vnitřní strukturou“ termů, ve VS Code napíšete tak,

[6] `https://github.com/madvorak/lean-mam/blob/main/mam/Cislo4.lean`

[7] Není náhodou, že používáme stejný symbol pro implikaci jako pro funkci. V typovém systému Leanu je totiž oboje to samé. Term typu `P → Q` je funkce, která transformuje term reprezentující důkaz `P` na term reprezentující důkaz `Q`. Díky tomu je také možné napsat spoustu důkazů (když je píšeme jako term; s taktikami je to malinko jiné) jednoduše pomocí skládání funkcí. Abyste napsali symbol ∘ pro složení funkcí, ve VS Code napište `\o` a stiskněte mezerník.

[8] Přesněji řečeno je unifikuje a hned splní „triviální“ podcíle.

[9] Ve skutečnosti je taktika `apply` ještě chytřejší. Máme-li předpoklad `(pqr : P → Q → R)` a cíl `R`, mohlo by se zdát, že `apply pqr` nebude fungovat, protože `pqr` má typ `P → (Q → R)` napsaný explicitně, kdežto cíl nemá typ `(Q → R)`. Jenže `apply pqr` zafunguje – vygeneruje dva nové cíle (`P` a `Q`) a uspokojí cíl `R`.

že napíšete `\<>` a stisknete tabulátor (nebo mezerník), což vloží začátek a konec špičaté závorky najednou. Taktika `constructor` na dalším řádku útočí na hlavní věc v cíli. Zde je tou hlavní věcí konjunkce. Vzniknou dva cíle místo původního jednoho cíle. První je dokázat `Q`. Druhý je dokázat `P`. Oba máme již v lokálním kontextu. Pro každý cíl použijeme taktiku `exact` se správným termem (zde jsou `p` a `q` jména, která jsme si zvolili výše). Jak jsme si ukázali minule, puntíky používáme pro přehlednost (byť zde asi není nic, v čem bychom se mohli poplést při čtení kódu).

Další logická spojka je disjunkce. Výraz `P ∨ Q` říká, že minimálně jedno z tvrzení `P` nebo `Q` musí platit. Ve VS Code napište `\or` a stiskněte mezerník. Toto je pravdivostní tabulka pro disjunkci:

P	Q	P ∨ Q
True	True	True
True	False	True
False	True	True
False	False	False

Pojďme si ukázat obdobné tvrzení o disjunkci (zde jsou místo puntíků svislítka, která znamenají něco jako „v případě tom“, ale jsou povinná):

```
example {P Q : Prop} (predpoklad : P ∨ Q) : Q ∨ P := by
  cases predpoklad with
  | inl p =>
    right
    exact p
  | inr q =>
    left
    exact q
```

Taktika `cases` rozbije `predpoklad` na možnosti, jak může být splněný. Dejte si pozor na správnou syntax – slovo `with` i svislítko před každou variantou jsou povinné. Názvy konstruktorů `inl` a `inr` vycházejí z definice disjunkce (není povoleno je pojmenovat jinak). Názvy jejich argumentů `p` a `q` si volíme my. Taktika `right` říká, že budeme splňovat cíl pomocí pravého podcíle (obecně podle druhého ze dvou konstruktorů). Použijeme `p` z lokálního kontextu. Taktika `left` říká, že budeme splňovat cíl pomocí levého podcíle (obecně podle prvního ze dvou konstruktorů). Použijeme `q` z lokálního kontextu.

Asi byste se zlobili, kdybychom opomenuli ekvivalenci. Výraz `P ↔ Q` říká, že `P` platí právě tehdy, když `Q` platí. Ve VS Code napište `\iff` a stiskněte mezerník. Toto je pravdivostní tabulka pro ekvivalenci:

P	Q	P ↔ Q
True	True	True
True	False	False
False	True	False
False	False	True

Ekvivalence je víc než jen syntaktická zkratka za konjunkci dvou navzájem obrácených implikací. Ekvivalenci můžeme používat k přepisu částí termů (v cíli nebo v předpokladu) pomocí `rw` a dalších taktik. Nejen že pravdivostní tabulka pro `P ↔ Q` je stejná jako pravdivostní tabulka pro `P = Q` nad patřičným typem, i jejich použití je skoro stejné!

Zatím jsme mluvili o binárních logických spojkách. Musíme zmínit také negaci, která je unární (tzn. bere jen jeden argument). Výraz `¬P` říká, že `P` neplatí. Ve VS Code napište `\not` a stiskněte tabulátor. Pravdivostní tabulka pro negaci je velmi jednoduchá:

P	¬ P
True	False
False	True

Negace je v Leanu implementována jako implikace. Podívejte se na pravdivostní tabulku pro implikaci a dosaďte si `Q = False`. Pouze dva řádky té tabulky budou relevantní. A uvidíte, že `¬P` a `(P → False)` jsou to samé. Syntaktický cukr jde ještě dál! Jsou-li `x` a `y` termy stejného typu, můžeme psát `x ≠ y` jako zkratku pro `¬(x = y)`, což je zkratka za `(x = y) → False`. Všechny tři výrazy jsou víc než jen logicky ekvivalentní – jsou si definičně rovny[10].

Úloha 4.1 [2b]: *Důkaz následujícího tvrzení napište pouze pomocí taktik 'exact', 'constructor', 'left', 'right', 'intro', 'use', 'cases', 'obtain'.*

```
example (P Q R : Prop) : P ∧ (Q ∨ R) ↔ (P ∧ Q) ∨ (P ∧ R) := by
  tauto
```

Predikátová logika

Pomocí výrokové logiky toho bohužel moc zajímavého nevyjádříme. Abychom mohli dělat zajímavou matiku, potřebujeme predikátovou logiku, která k uvedeným logickým spojkám přidává kvantifikátory.

Máme k dispozici univerzální kvantifikátor ∀ ve významu „pro každý term daného typu“ a existenční kvantifikátor ∃ ve významu „pro nějaký term daného typu“. VS Code zkratky pro jejich zápis `\all` a `\ex` vycházejí z angličtiny („all“ a „exists“).

Predikátovou logiku jsme již implicitně používali v minulém čísle – argumenty v závorce označují proměnné, před které se automaticky vkládá univerzální kvantifikátor (a zároveň se pro důkaz daného tvrzení automaticky přidává tato proměnná do kontextu). V minulém čísle jsme psali:

```
theorem dva_krat (n : ℕ) : 2 * n = n + n := two_mul n
```

Ve skutečnosti to znamenalo:

```
theorem dva_krat : ∀ n : ℕ, 2 * n = n + n := fun n => two_mul n
```

[10] V Leanu existuje víc typů rovnosti. Definiční rovnost je jedna z těch silnějších.

To samé lze jednodušeji napsat takto:

```
theorem dva_krat : ∀ n : ℕ, 2 * n = n + n := two_mul
```

Zde je celý typ termu `dva_krat` napsaný za hlavní dvojtečkou (zde tou první dvojtečkou).

Jak se tvrzení s kvantifikátory dokazuje? Pokud cíl začíná ∀ něco, chápeme to, že nepřítel vybírá, pro jakou hodnotu musíme poskytnout důkaz. Taktika `intro` si přečte hodnotu od nepřítele – do kontextu je přidán nový identifikátor, o kterém zatím není známo nic kromě jeho typu. Pokud cíl začíná ∃ něco, je na nás, abychom si vybrali, pro jakou hodnotu budeme zbytek tvrzení dokazovat. Taktika `use` následovaná termem správného typu eliminuje existenční kvantifikátor.

Pokud je kvantifikátor v předpokladu, je situace skoro opačná. Protože tu nemáme prostor vysvětlovat všechny důležité taktiky, odkážeme se na jejich přehled na webu[11].

Práci s kvantifikátory si ukážeme na tomto jednoduchém příkladu:

```
theorem deMorgan_existencni {α : Type} {R : α → Prop}
    (R_lze_splnit : ∃ a : α, R a) : ¬ (∀ a : α, ¬ R a) := by
  obtain ⟨a, splnen⟩ := R_lze_splnit
  intro pro_spor
  apply pro_spor
  exact splnen
```

Řádek `obtain ⟨a, splnen⟩ := R_lze_splnit` nám uloží svědka pod jménem `a` a důkaz `(R a)` pod jménem `splnen`. Cíl začíná negací. Formálně má cíl tvar funkce `(∀ a : α, ¬ R a) → False`, takže na něj použijeme taktiku `intro` jako u implikací. V lokálním kontextu se objeví `pro_spor : (∀ a : α, ¬ R a)` a cílem je teď `False`. To se krásně unifikuje s pravou stranou `pro_spor`. Proto použijeme taktiku `apply`. Poslední řádek vyplývá z nového cíle.

Alternativní přístupy k tomuto důkazu najdete v našem repozitáři. Najdete tam důkaz využívající taktiku `push_neg`, automatický důkaz taktikou `tauto`, a nakonec důkaz nalezený pomocí `library_search`. Taktika `tauto` je pro důkazy základních logických tvrzení tak silná, že ji máte v prvních dvou úlohách zakázanou.

Všimněte si závorek v hlavičce. Jak jsme si už vysvětlili, složená závorka označuje implicitní argument (automaticky se odvodí). Kulatá závorka ukazuje jediný explicitní argument (ten musíme poskytnout při zavolání). Kdykoliv poskytneme term `(R_lze_splnit : ∃ a : α, R a)`, z jeho typu se přímočaře odvodí α i `R`, ze kterých se posléze odvodí typ výstupu (jaké přesně tvrzení to konkrétní volání `deMorgan_existencni` poskytne).

Podívejte se i na jiné příklady v našem repozitáři. Naučíte se z nich syntax Leanu a různé důkazové techniky (zdaleka ne všechno se vešlo do tohoto časopisu).

[11] `https://github.com/madvorak/lean4-tactics/blob/main/README-CZE.md`

Úloha 4.2 [2b]: *Důkaz následujícího tvrzení napište pouze pomocí taktik 'exact', 'constructor', 'left', 'right', 'intro', 'use', 'cases', 'obtain'.*

```
example (R : ℕ → ℕ → ℕ → ℕ → ℕ → Prop) :
    (∃ x, ∀ y, ∀ z, ∀ b, ∃ a, R a b x y z) →
    (∀ z, ∀ y, ∃ x, ∀ w, ∃ v, R v w x y z) := by
  tauto
```

Množiny a funkce

Často se říká, že celá matika stojí na teorii množin. Lean však na teorii množin nestojí. Lean je založen na teorii typů (konkrétně na teorii závislých typů nazvané Calculus of Inductive Constructions). I v teorii typů lze samozřejmě mluvit o množinách, ale nejsou tu tím nejzákladnějším pojmem. Jsou dva způsoby jejich reprezentace. Neformálně je nazveme „množina je typ" a „množina je predikát".

Pohled „množina je typ" je velice přímočarý. Máme určitý typ, o kterém prohlásíme, že na všechny termy toho typu se budeme dívat tak, že tvoří množinu. Pokud bychom chtěli zadefinovat nějakou podmnožinu této množiny, můžeme použít podtyp. Tenhle přístup se však rychle stane zbytečně technicky složitým. Nebudeme se tu učit deklaraci vlastních typů a už vůbec ne deklaraci podtypů.

Pohled „množina je (unární) predikát" je vhodný pro mluvení o podmnožinách. Když chceme mluvit o množině `M`, která obsahuje nějaká reálná čísla (je jedno, zda je tato množina konečná či nekonečná), zadeklarujeme si `(M : Set ℝ)`, což je syntaktická zkratka za `(M : ℝ → Prop)`. Když chceme mluvit o množině `abeceda`, která obsahuje nějaké znaky, zadeklarujeme si `(abeceda : Set Char)`, což je syntaktická zkratka za `(abeceda : Char → Prop)`.

Mějme dvě množiny `X` a `Y` obsahující nějaká celá čísla, tj. `(X Y : Set ℤ)`. Pokud chceme říct, že `X` je podmnožina `Y`, stačí napsat `X ⊆ Y`, kde infixový operátor ⊆ zapíšeme zkratkou `\ss` ve VS Code, což se přeloží na term `Set.Subset X Y` neboli `X.Subset Y` definovaný takto:

```
∀ a, a ∈ X → a ∈ Y
```

Tohle je stále ještě syntaktický cukr. Plná verze `X ⊆ Y` je ve skutečnosti toto:

```
∀ a : ℤ, X a → Y a
```

Lean s knihovnou Mathlib nám dávají většinu notace pro práci s množinami, kterou známe z matematiky. Pokud bychom například chtěli zapsat (pravdivé) tvrzení, že množina celých čísel větších či rovných stu je podmnožinou kladných celých čísel, můžeme to napsat takto:

```
{ x : ℤ | x ≥ 100 } ⊆ { y : ℤ | y > 0 }
```

Identifikátory `x` a `y` jsou arbitrární; zvolili jsme je jako náznak jmen našich množin `X` a `Y`.

V praxi oba uvedené pohledy kombinujeme; pohled „množina je typ“ používáme pro hlavní množinu; pohled „množina je predikát“ používáme pro podmnožiny. Není to jediná možnost – množinu všech celých čísel bychom mohli psát `{ z : ℤ | True }` ve stylu „množina je predikát“, ale připadá nám jednodušší napsat prostě ℤ ve stylu „množina je typ“. Pozor na to, že ty dva termy mají odlišný typ. Nelze vyslovit rovnost `ℤ = { z : ℤ | True }`.

Pokud vám připadá celé to povídání o množinách zbytečně zdlouhavé, můžete si místo něj odnést zjednodušené pravidlo, které vyjadřuje, jak s množinami pracujeme my (i většina ostatních uživatelů Leanu):

- množina = `Type`
- podmnožina = `Set`

Tohle rozhodně nepište do písemky z matematiky ani do písemky z angličtiny, ale jinak budete po většinu času v pohodě, když si to takhle zapamatujete.

Ukážeme si ještě nějaký nepovinný cukřík – kvantifikace přes (pod)množinu. Z neformální matematiky znáte tvrzení typu „pro každý prvek M platí $(\dots)$“ či „existuje prvek M takový, že $(\dots)$“. Podobnou věc můžeme psát i v Leanu! Mějme v kontextu `(M : Set ℝ)`. Teď lze psát o množině `M` věci jako:

```
∃ a ∈ M, a ^ 4 ≤ 10
```

To se přeloží (s využitím údaje o typu `M`) na:

```
∃ a : ℝ, a ∈ M ∧ a ^ 4 ≤ 10
```

Když spojíte doposud nabyté znalosti dohromady, mělo by vám dávat smysl, že poslední uvedené tvrzení vyjadřuje to samé, co by intuitivně mělo znamenat to tvrzení nad ním, čili že množina `M` obsahuje nějaký prvek, jehož čtvrtá mocnina je menší či rovna deseti. Pravdivost tohoto tvrzení samozřejmě závisí na tom, jaké prvky `M` obsahuje. Obdobně můžeme psát o množině `M` věci jako:

```
∀ a ∈ M, a ≤ |a|
```

To se přeloží (s využitím údaje o typu `M`) na:

```
∀ a : ℝ, a ∈ M → a ≤ |a|
```

Když použijeme ještě méně syntaktického cukru, je to tohle tvrzení:

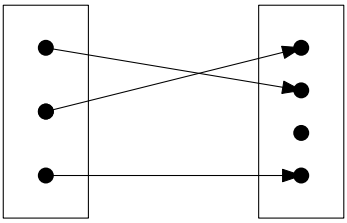
```
∀ a : ℝ, M a → a ≤ abs a
```

Zrovna tohle tvrzení platí bez ohledu na to, jaké prvky `M` obsahuje. Můžete si ho zkusit dokázat (nebo se podívat do repozitáře – jsou to dva nudné řádky).

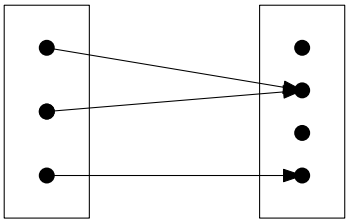
Zavedeme si několik pojmů o funkcích (neboli zobrazeních). Pak si o nich dokážeme pár vlastností.

První pojem nejspíš znáte ze školy; funkce je prostá, pokud dva různé prvky vždy zobrazí na dva různé obrazy:

```
def Prosta {A B : Type} (f : A → B) : Prop :=
  ∀ x y : A, x ≠ y → f x ≠ f y
```



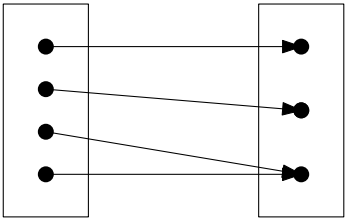
je prostá



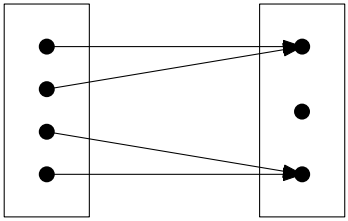
není prostá

Funkce je surjektivní (neboli „funkce na“), pokud každý prvek cílové množiny má vzor:

```
def Surjektivni {A B : Type} (f : A → B) : Prop :=
  ∀ z : B, ∃ x : A, f x = z
```



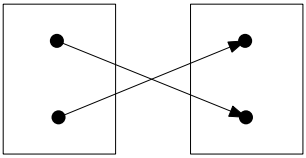
je surjektivní



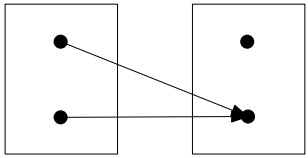
není surjektivní

Funkce je bijektivní (neboli bijekce), pokud je prostá a surjektivní zároveň:

```
def Bijektivni {A B : Type} (f : A → B) : Prop :=
  Prosta f ∧ Surjektivni f
```



je bijektivní



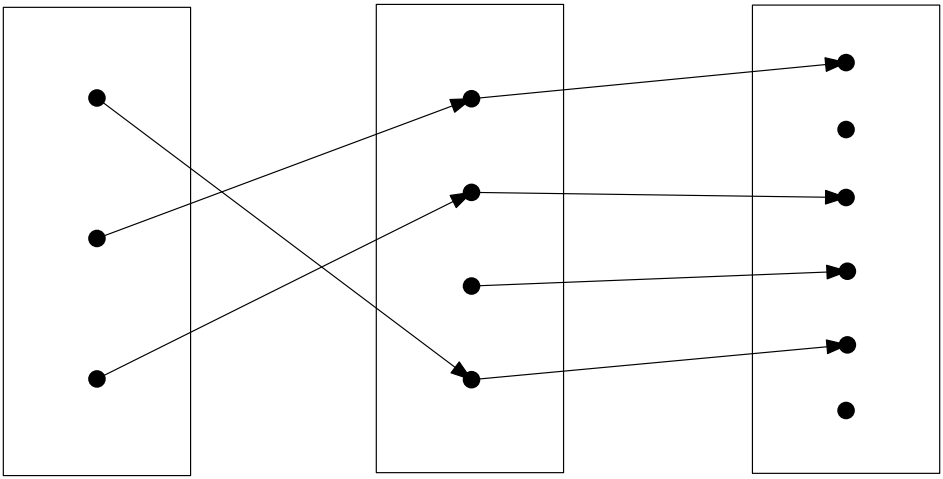
není bijektivní

Dvě množiny (nebo dva typy) jsou stejně velké, pokud mezi nimi existuje bijektivní funkce:

```
def StejneVelke (A B : Type) : Prop := ∃ f : A → B, Bijektivni f
```

Dokažme si teď velice přirozené tvrzení, že složením prostých funkcí vznikne prostá funkce:

```
theorem slozProsta {A B C : Type} {f : A → B} {g : B → C}
    (hf : Prosta f) (hg : Prosta g) :
    Prosta (g ∘ f) := by
  sorry
```



Protože cílem je dokázat `Prosta (g ∘ f)`, podíváme se nejprve, jak je `Prosta` definována (v našem kódu je to jen několik řádků nad tím, ale často to tak být nemusí; hodí se vědět, že se k definici můžeme rychle dostat tak, že klikneme na jméno a stiskneme F12, což je jedna z kláves v horním řádku klávesnice).

Začíná to univerzálním kvantifikátorem. Takže napíšeme `intro x` a přečteme si nový cíl:

$$\forall\ \texttt{y : A, x} \neq \texttt{y} \to \texttt{(g} \circ \texttt{f) x} \neq \texttt{(g} \circ \texttt{f) y}$$

Jasným pokračováním je `intro y`. V lokálním kontextu přibyde další položka a cíl vypadá takto:

$$\texttt{x} \neq \texttt{y} \to \texttt{(g} \circ \texttt{f) x} \neq \texttt{(g} \circ \texttt{f) y}$$

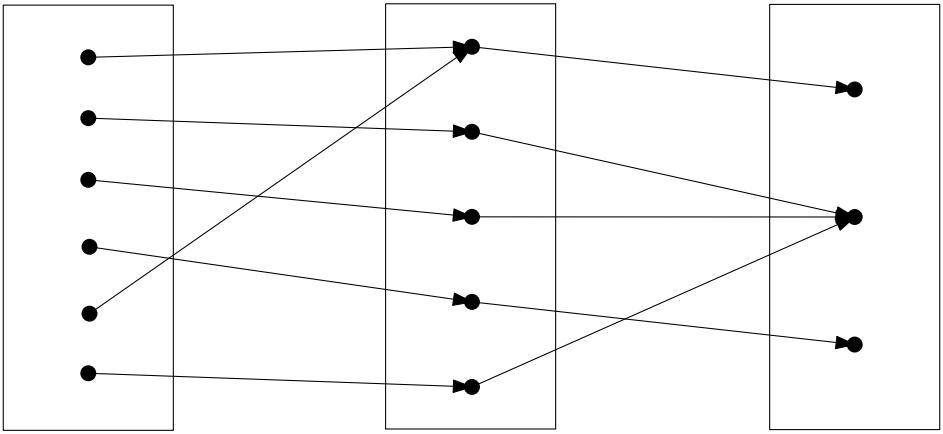
Je to implikace, takže taktiku `intro` použijeme ještě jednou. Všechny tři můžeme napsat na jeden řádek `intro x y hxy`, načež je cílem dokázat `(g ∘ f) x ≠ (g ∘ f) y` neboli `g (f x) ≠ g (f y)`. To krásně pasuje na pravou stranu tvrzení `hg (f x) (f y)`. Dalším krokem je proto `apply hg`. Zbytek důkazu je už jasný. Takhle vypadá celý důkaz tvrzení, že složení prostých funkcí dá prostou funkci:

```
theorem slozProsta {A B C : Type} {f : A → B} {g : B → C}
    (hf : Prosta f) (hg : Prosta g) :
    Prosta (g ∘ f) := by
  intro x y hxy
  apply hg
  apply hf
  exact hxy
```

Ve všech úlohách od teď můžete používat jakékoliv taktiky.

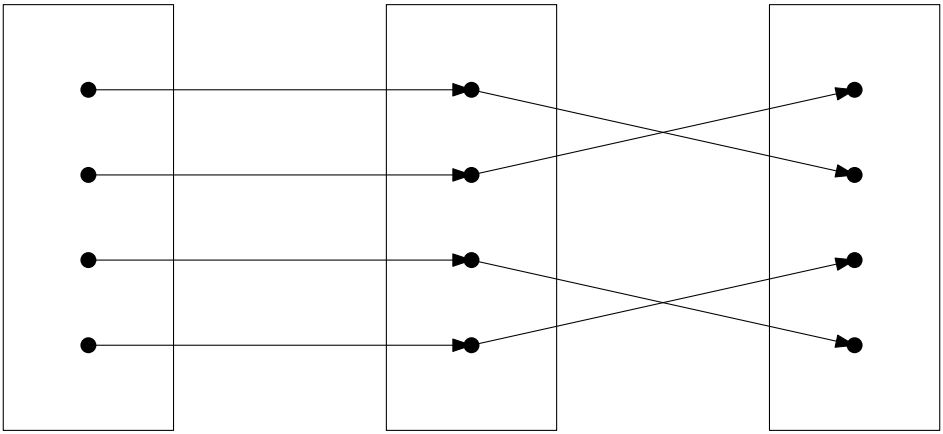
Úloha 4.3 [1b]: *Dokažte, že složení surjektivních funkcí dá surjektivní funkci:*

```
theorem slozSurjektivni {A B C : Type} {f : A → B} {g : B → C}
    (hf : Surjektivni f) (hg : Surjektivni g) :
    Surjektivni (g ∘ f) := by
  sorry
```



Úloha 4.4 [1b]: *Dokažte, že složení bijektivních funkcí dá bijektivní funkci:*

```
theorem slozBijektivni {A B C : Type} {f : A → B} {g : B → C}
    (hf : Bijektivni f) (hg : Bijektivni g) :
    Bijektivni (g ∘ f) := by
  sorry
```



Pojďme si ukázat víc tvrzení o množinách a zobrazeních mezi nimi! V předloňském tématku Nekonečna jsme narazili na Cantorovu větu[12]. Viděli jsme obrázek, který objasňoval, proč je nemožné přirozenými čísly očíslovat množiny přirozených čísel.

Když to zobecníme na jakékoliv množiny (platí to pro konečné i nekonečné množiny, ale překvapivé je to jen u nekonečných množin), můžeme tvrdit, že neexistuje žádná surjektivní funkce z množiny do její potenční množiny. V Leanu mluvíme o funkci typu `T → Set T`, tedy o zobrazení, které prvkům typu `T` přiřazuje množiny nad typem `T`, neboli prvkům množiny `T` přiřazuje podmnožiny `T`. Taková zobrazení jistě existují (třeba každému prvku přiřaďme jednoprvkovou množinu – uživatelé Leanu je nazývají „singleton“), ale žádné z nich není surjektivní. Proč? To nám ukáže Cantorova diagonální metoda!

Pojďme si Cantorovu větu dokázat formálně... Vysvětlení si můžete přečíst v odstavcích pod kódem důkazu.

```
theorem vetaCantor (T : Type) :
    ¬ (∃ f : T → Set T, Surjektivni f) := by
  intro pro_spor
  obtain ⟨f, surjektivni⟩ := pro_spor
  obtain ⟨a, sporne⟩ := surjektivni { x : T | x ∉ f x }
  have paradox : (a ∈ f a) ↔ (a ∉ f a)
  • exact of_eq (congr_arg (Membership.mem a) sporne)
  exact nemozna_ekvivalence paradox
```

Řádek začínající puntíkem byl vygenerován pomocí `library_search`. Ostatní řádky byly napsány ručně.

Zaměřme se na stav důkazu, který vidíme v Infoview po prvních dvou krocích důkazu. Lokální kontext má pouze tři položky:

```
T : Type
f : T → Set T
surjektivni : Surjektivni f
```

Cílem je dokázat `False`.

Tvrzení `surjektivni` musíme nějak dovést ke sporu. Naštěstí víme jak na to! Nahlížejme na `surjektivni` jako na funkci, která libovolnému termu typu `Set T` přiřadí term typu `T`, jež by měl být jeho vzorem. Inu, dosadíme tam (a to je nejtěžší krok tohoto důkazu) množinu `{ x : T | x ∉ f x }`. Cílem je teď stále dokázat `False`, ale v lokálním kontextu nám přibyly dvě nové položky navíc (přesné znění se může lišit dle verze Leanu a knihoven):

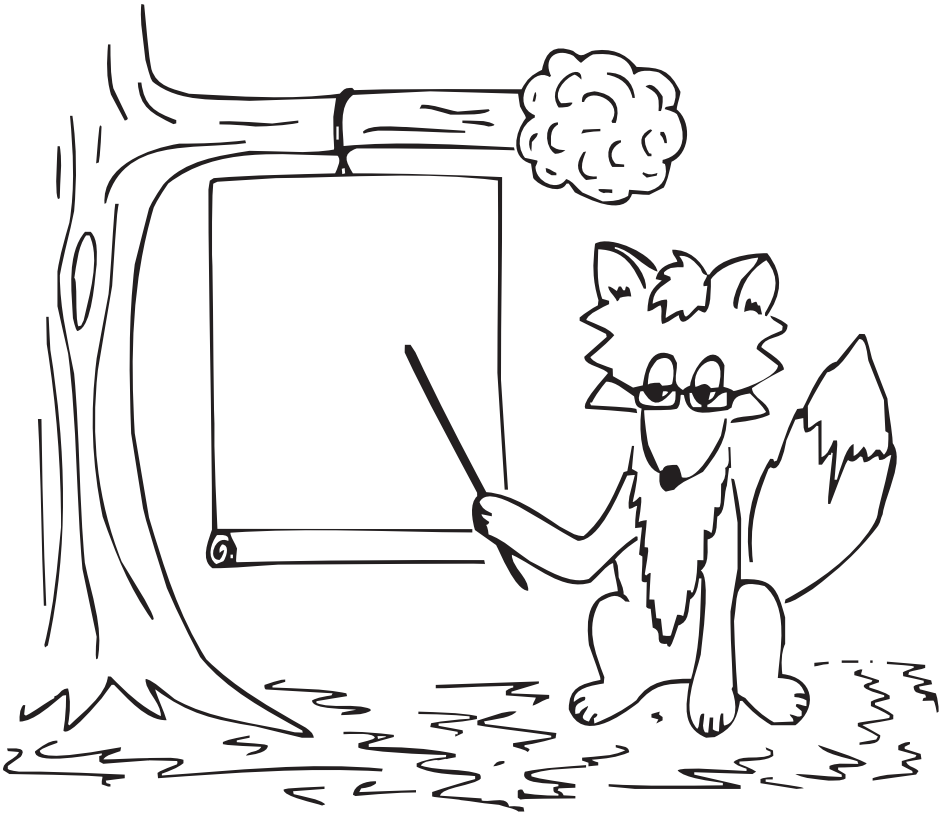
```
a : T
sporne : f a = { x : T | x ∉ f x }
```

Obsahuje množina `f a` prvek `a`? Ano i ne!

[12] `https://mam.mff.cuni.cz/media/cislo/pdf/28/28-1.pdf#page=27` předposlední strana

Přímo z definice máme, že prvek náleží do množiny, pokud pro něj platí, co je napsáno vpravo od svislítka. Pro prvek jménem `a` tou charakterizací je `a ∉ f a`. Ale co že jsme to právě charakterizovali? Zapisovali jsme podmínku pro `a ∈ f a`. Dospěli jsme k závěru `(a ∈ f a) ↔ (a ∉ f a)`. Tadá!

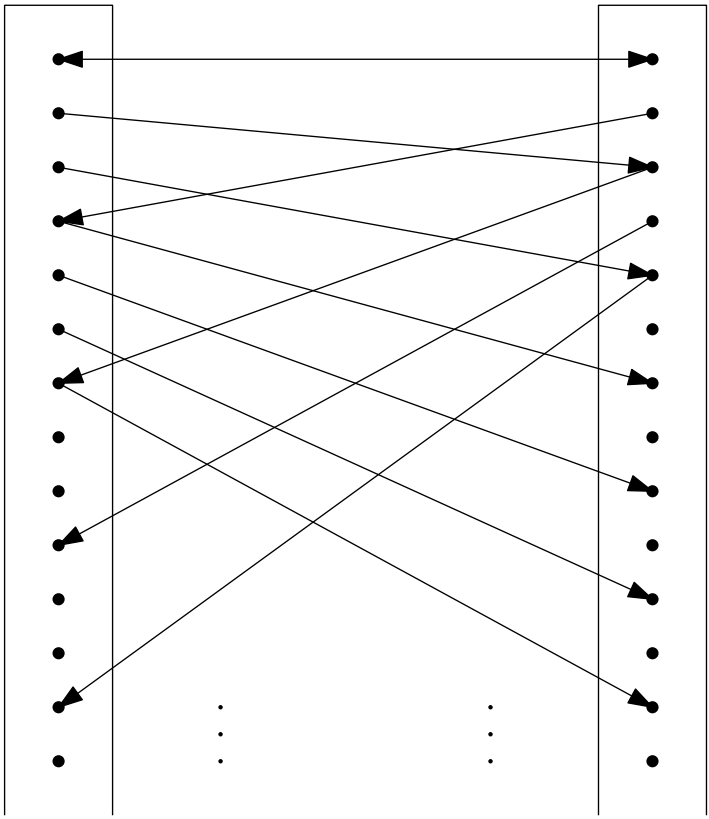
Poslední řádek (konstrukce termu typu `False` z termu `paradox`) lze nahradit zavoláním taktiky `tauto`.



Nyní vyslovíme významnou matematickou větu, která nám umožní charakterizovat „stejnovelikost“ bez nutnosti konstruovat konkrétní bijekci:

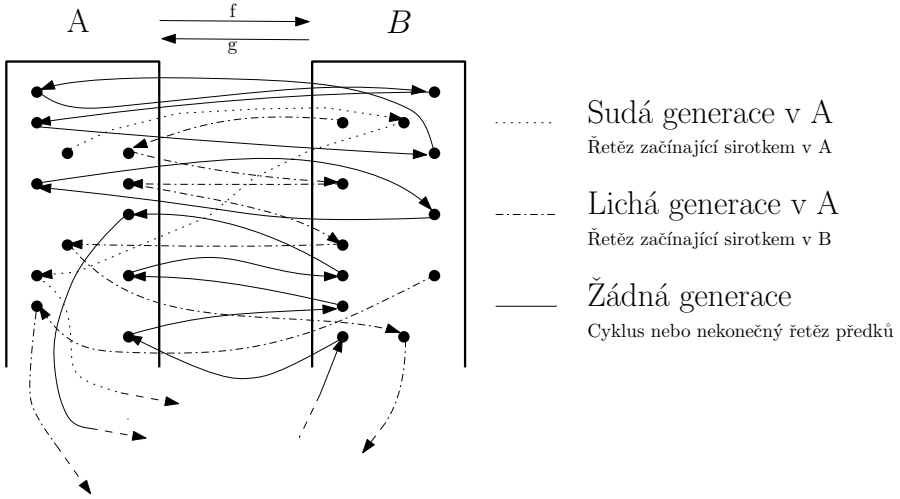
```
theorem jsouStejneVelke {A B : Type} :
    (∃ f : A → B, Prosta f) ∧ (∃ g : B → A, Prosta g) →
      StejneVelke A B := by
  sorry
```

Nejprve nahlédněme, že pro konečné množiny je to triviální. Máme-li mezi konečnými množinami `A` a `B` jak prostou funkci `f : A → B`, tak i prostou funkci `g : B → A`, pak obě tyto funkce jsou nutně bijektivní (a funkci `f` můžeme hned použít jako svědka, že jsou množiny `A` a `B` stejně velké). Pro nekonečné množiny to už tak snadné není...



Na obrázku jsou dvě nekonečné množiny a dvě prosté funkce, které nejsou bijektivní. Vidíme tedy, že když to chceme dokázat obecně, bude potřeba nějaký trik...

Potřebujeme zkonstruovat bijekci `F : A → B` pouze na základě prosté funkce `f : A → B` a prosté funkce `g : B → A`. Problém s nekonečnými množinami je, že existují prvky `A`, které nemají `g`-předchůdce v `B`, a že existují prvky `B`, které nemají `f`-předchůdce v `A`. Nazvěme tyto prvky sirotky neboli nultou generací. Mohli bychom si říct, že funkci `f : A → B` rozšíříme tím, že pro každého sirotka `b : B` necháme `F` zobrazit `g b` na `b`, čili tak trochu využijeme g^{-1} v konstrukci `F`. Bude to fungovat? Máme tu skupinu prvků `A`, kterou nazveme první generací, kterou necháme jít podle funkce g^{-1} namísto funkce `f`. Tím ale vzniká skupina `f`-obrazů v `B`, kterou nazveme druhou generací, která by neměla `F`-předchůdce,

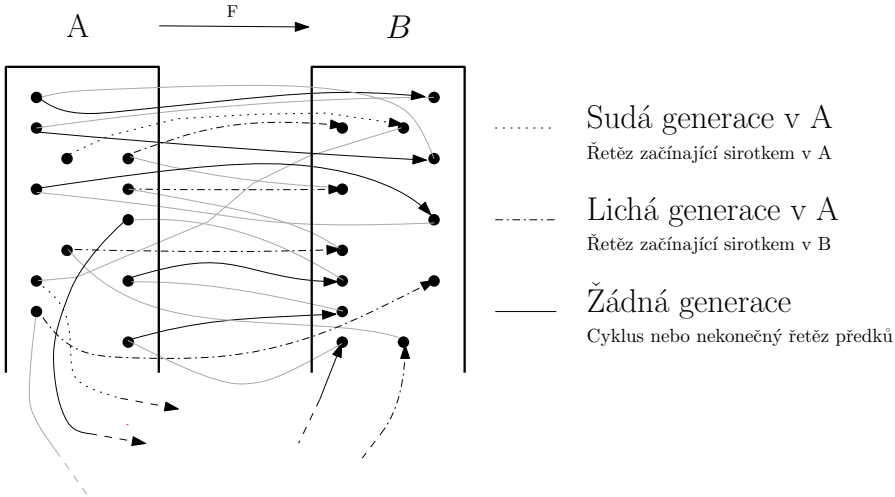


což se nesmí stát. Záplatou na tento problém bude, že třetí generaci prvků v `A` zase zobrazíme pomocí $\mathtt{g}^{-1}$. Tím ztratí své předchůdce čtvrtá generace prvků v `B`, kvůli čemuž i na pátou generaci prvků v `A` použijeme funkci $\mathtt{g}^{-1}$. Asi už vidíte, kam to vede. Máme prvky liché generace v `A`, kde si přejeme, aby `F` opisovala $\mathtt{g}^{-1}$, a prvky sudé generace v `A`, kde si přejeme, aby `F` kopírovala `f`. Pak ještě existují prvky, které se celému tomu žonglování vyhnuly, poněvadž, když od nich půjdeme libovolně dlouho proti směru šipek, nikdy nedorazíme k sirotkovi v `A` ani k sirotkovi v `B`. Tyto prvky, které leží v cyklech nebo v oboustranně-nekonečných řetězech, nám naštěstí naši konstrukci `F` nerozbijí. Použijeme na ně funkci `f`.

Abychom ten dlouhý odstavec shrnuli, konstruujeme funkci `F : A → B`, která se na prvcích liché generace v `A` chová jako $\mathtt{g}^{-1}$ a na všech ostatních prvcích `A` se (sudé generace, cykly, oboustranně-nekonečné řetězy) chová jako `f`. Zbývá dokázat, že tato funkce `F` je bijektivní. Tím bude hotov náš důkaz, že množiny `A` a `B` jsou stejně velké.

Pojďme napsat formální důkaz! Následuje několik pomocných definicí a lemmat. Některá dokážeme my. Některá dokážete vy. Také dokončení důkazu hlavní věty bude na vás!

```
inductive Generace : {A B : Type} → (A → B) → (B → A) →
    A → ℕ → Prop
| nula {A B : Type} {f : A → B} {g : B → A} {a : A}
    (sirot : ¬ ∃ b : B, g b = a) :
    Generace f g a 0
| nasl {A B : Type} {f : A → B} {g : B → A} {a : A} (p : B)
    (rodic : g p = a) {n : ℕ} (rodokmen : Generace g f p n) :
    Generace f g a n.succ
```



`Generace` je definována induktivně. Nemusíte se ale učit, jak se píší induktivní definice. Bude vám stačit vědět, že když potřebujete dokázat náležení do určité generace, můžete použít taktiku `left` následovanou důkazem `(sirot : ¬ ∃ b : B, g b = a)` nebo taktiku `right` následovanou důkazy `(rodic : g p = a)` a `(rodokmen : Generace g f p n)`. Naopak když je `Generace` v předpokladu, tak taktikou `cases` musíte obsloužit jeden či oba možné případy (ona si řekne).

Nyní následují definice, co je to sudá generace a lichá generace. Typové argumenty `A` a `B` jsou od `variable` dál vkládány jako implicitní do všech deklarací.

```
variable {A B : Type}

def SudaGenerace (f : A → B) (g : B → A) (a : A) : Prop :=
  ∃ n : ℕ, Generace f g a (2*n)

def LichaGenerace (f : A → B) (g : B → A) (a : A) : Prop :=
  ∃ n : ℕ, Generace f g a (2*n + 1)
```

Nyní se objeví nové klíčové slovo `lemma`. Znamená to samé jako `theorem`. Rozdíl v nich je pouze pro člověka – když čtenář narazí na tvrzení uvozené slovem `lemma`, dá mu to indikaci, že toto tvrzení nemá užitek samo o sobě, ale jen jako nástroj na důkaz jiného tvrzení. Zde budeme mít řadu `lemma`t, jejichž jediným účelem bude, že nám pomůžou dokázat `theorem jsouStejneVelke` na konci.

Úloha 4.5 [1b]: *Dokažte, že každý prvek s lichým počtem předchůdců má nějakého rodiče a ten má sudý počet předchůdců:*

```
lemma LichaGenerace.existuje_rodic {f : A → B} {g : B → A}
    {a : A} (lichaGen : LichaGenerace f g a) :
    ∃ p : B, g p = a ∧ SudaGenerace g f p := by
  sorry
```

Úloha 4.6 [2b]: *Dokažte, že pokud prvek má lichý počet předchůdců, jeho potomek má sudý počet předchůdců:*

```
lemma LichaGenerace.pristiSudaGenerace {f : A → B} {g : B → A}
    {a : A} (lichaGen : LichaGenerace f g a) :
    SudaGenerace g f (f a) := by
  sorry
```

Úloha 4.7 [2b]: *Dokažte, že pokud prvek má sudý počet předchůdců, jeho potomek má lichý počet předchůdců:*

```
lemma SudaGenerace.pristiLichaGenerace {f : A → B} {g : B → A}
    {a : A} (sudaGen : SudaGenerace f g a) :
    LichaGenerace g f (f a) := by
  sorry
```

Úloha 4.8 [3b]: *Dokažte, že pokud nesirotek má sudý počet předchůdců, jeho rodič má lichý počet předchůdců:*

```
lemma SudaGenerace.predchoziLichaGenerace {f : A → B} {g : B → A}
    {a : A} (sudaGen : SudaGenerace g f (f a)) (hf : Prosta f) :
    LichaGenerace f g a := by
  sorry
```

Budeme potřebovat inverzi k funkci `g : B → A`. Nemůžeme však očekávat existenci funkce typu `A → B` inverzní ke `g`.

```
noncomputable def inverze {f : A → B} {g : B → A}
    {a : A} (lichaGen : LichaGenerace f g a) : B :=
  lichaGen.existuje_rodic.choose
```

Pro jednoduchost jsme funkci g^{-1} nedefinovali na celém oboru hodnot funkce `g`, nýbrž jen pro prvky s lichým počtem předchůdců. Tato část funkce `g` bude užitečná pro konstrukci bijekce `F : A → B`.

V definici vás možná zaujalo použití teček podobné syntaxi objektově orientovaných jazyků. Máme-li deklaraci jménem `LichaGenerace.existuje_rodic` a máme-li argument `(lichaGen : LichaGenerace f g a)`, můžeme namísto `LichaGenerace.existuje_rodic lichaGen` napsat `lichaGen.existuje_rodic` pro vyšší čitelnost a estetiku. Aby to fungovalo, musí mít první argument stejný typ, jako je část názvu stojící před tečkou. Následně je tečka použita ještě jednou. Jedná se o zápis zavolání funkce `Exists.choose` na argument, jehož typ začíná existenčním kvantifikátorem. Výsledný term `lichaGen.existuje_rodic.choose` je čitelnější než `Exists.choose (LichaGenerace.existuje_rodic lichaGen)`.

Funkce `Exists.choose` zpřístupňuje tzv. axiom výběru[13]. Nechceme jít tak hluboko, abychom tento axiom vysvětlovali. Doporučujeme vám neřešit, co se děje na pozadí, a jednoduše akceptovat, že klíčové slovo `noncomputable` je zde nutností. Následující dvě lemmata, která s axiomem výběru přímo pracují, dostanete dokázaná:

```
lemma LichaGenerace.g_inverze {f : A → B} {g : B → A}
    {a : A} (lichaGen : LichaGenerace f g a) :
    g (inverze lichaGen) = a :=
  lichaGen.existuje_rodic.choose_spec.left

lemma LichaGenerace.sudaGen_inverze {f : A → B} {g : B → A}
    {a : A} (lichaGen : LichaGenerace f g a) :
    SudaGenerace g f (inverze lichaGen) :=
  lichaGen.existuje_rodic.choose_spec.right
```

Úloha 4.9 [2b]: *Dokažte, že funkce* $\mathtt{g}^{-1}$ *se tam, kde je definována, chová jako prostá funkce:*

```
lemma inverze_jakoProsta {f : A → B} {g : B → A}
    {x y : A} (hxy : x ≠ y)
    (hx : LichaGenerace f g x) (hy : LichaGenerace f g y) :
    inverze hx ≠ inverze hy := by
  sorry
```

Úloha 4.10 [10b]: *Dokončete tento důkaz:*

```
theorem jsouStejneVelke :
    (∃ f : A → B, Prosta f) ∧ (∃ g : B → A, Prosta g) →
      StejneVelke A B := by
  intro ⟨⟨f, hf⟩, ⟨g, hg⟩⟩
  classical
  let F : A → B := fun a =>
      if ha : LichaGenerace f g a then inverze ha else f a
  sorry
```

Ufff. Vzpomínáte si, jak jsme na začátku tématka řekli tohle?

„To, že tématko nevyžaduje předchozí znalosti, však neznamená, že tématko bude snadné. Uvidíte, že obtížnost úloh bude podstatně narůstat.“

Tak tohle jsme tím mysleli.

[13] https://github.com/leanprover/lean4/blob/b614ff1d12bc38f39077f9ce9f2d48b42c003ad0/src/Init/Prelude.lean#L704-L725

Řešení úloh z 3. dílu

Zadání:
Pomocí `library_search` *najděte důkaz zadaného tvrzení.*

Řešení:

```
theorem na_druhou (a : ℚ) : a ^ 2 = a * a := pow_two a
```

Zadání:

```
example (a b c d : ℕ)
    (a_je : a = b + d) (b_je : b = a * a)
    (c_je : c = b + d) (d_je : d = c * c) :
    b ^ d = d ^ b := by
```

Řešení:
Nejprve snadno nahlédneme `c = a` pomocí:

```
  rw [←a_je] at c_je
```

Poté získáme `d = b` takto:

```
  rw [c_je, ←b_je] at d_je
```

Pro dokončení důkazu převedeme obě strany cíle na `b ^ b` jediným přepsáním:

```
  rw [d_je]
```

Zadání:

```
example (a b c d : ℤ)
    (a_je : a = d ^ 4) (b_je : b = 1 / c)
    (c_je : c = a - b) (d_je : d = 4 * a) :
    (a + b) ^ 2 - c ^ 2 = b * d := by
```

Řešení:
Nejprve si uvědomíme, že použití předpokladů `a_je` a `b_je` by pouze komplikovalo řešení. Poté použijeme předpoklady `c_je` a `d_je` takto:

```
  rw [c_je, d_je]
```

Zbývá dokázat cíl:

```
(a + b) ^ 2 - (a - b) ^ 2 = b * (4 * a)
```

To je jen několik algebraických manipulací. Důkaz toho nemusíme psát ručně, protože na to máme taktiku:

```
  ring
```

Zadání:

```
example (x : ℝ) :
    50*x^2 - 126*x + 96 ≥ 0 := by
```

Řešení:
Identifikujeme čtverec, po jehož odečtení zbydou jen triviálně nezáporné členy. Zbytek důkazu je mechanický.

```
  have : 49*x^2 - 126*x + 81 ≥ 0
  • convert_to (7*x - 9) ^ 2 ≥ 0
    • ring
    exact pow_two_nonneg (7*x - 9)
  nlinarith
```

Zadání:

```
example (x y : ℝ) :
    2 * x^3 * y^3 ≤ x^4 * y^2 + x^2 * y^4 := by
```

Řešení:
Hodíme všechno na stejnou stranu. Pak už je to čtverec.

```
  have : 0 ≤ x^4 * y^2 + x^2 * y^4 - 2 * x^3 * y^3
  • convert_to 0 ≤ (x^2*y - x*y^2) ^ 2
    • ring
    exact pow_two_nonneg (x^2*y - x*y^2)
  linarith
```

Zadání:

```
example (x y z : ℝ) :
    4*x^2 + 12*x*y - 4*x*z + 9*y^2 - 6*y*z + z^2 ≥ 0 := by
```

Řešení:
Tady je to celé jeden čtverec.

```
  convert_to (2*x + 3*y - z) ^ 2 ≥ 0
  • ring
  exact pow_two_nonneg (2*x + 3*y - z)
```

Zadání:

```
example (a b : ℝ) (ha : 0 < a) (hb : 0 < b) :
    1 / a + 1 / b ≤ a / b^2 + b / a^2 :=
  sorry
```

Řešení:
Měli jsme připraveno jiné vzorové řešení, pak ale Mgr.^{MM} Barbora Vosáhlová přišla s elegantnějším nápadem. Uvedeme tu proto její řešení (pouze nepatrně upraveno):

Řešení od Mgr.^{MM} Barbory Vosáhlové:

```
example (a b : ℝ) (ha : 0 < a) (hb : 0 < b) :
    1 / a + 1 / b ≤ a / b^2 + b / a^2 := by
  have : (a - b) ^ 2 * (a + b) ≥ 0
  • nlinarith
  have : a^3 + b^3 - a*b^2 - a^2*b ≥ 0
  • convert this
    ring
  have : a^3 + b^3 ≥ a*b^2 + a^2*b
  • linarith
  have : (a^3 + b^3) / (a^2 * b^2) ≥ (a*b^2 + a^2*b) / (a^2 * b^2)
  • have levy_citatel_nezap : 0 ≤ a^3 + b^3
    • nlinarith
    have jmenovatel_klad : 0 < a^2 * b^2
    • exact mul_pos (sq_pos_of_pos ha) (sq_pos_of_pos hb)
    have samozr : a^2 * b^2 ≤ a^2 * b^2
    • rfl
    exact div_le_div levy_citatel_nezap this jmenovatel_klad samozr
  have : a^3 / (a^2 * b^2) + b^3 / (a^2 * b^2) ≥
         (a * b^2) / (a^2 * b^2) + (a^2 * b) / (a^2 * b^2)
  • convert this
    • exact div_add_div_same (a^3) (b^3) (a^2 * b^2)
    ring
  convert this using 2 <;>
    field_simp [ne_of_gt ha, ne_of_gt hb] <;> ring
```

Syntax `<;>` znamená „aplikuj následující taktiku na všechny vzniklé cíle“.

Martin Dvořák; `martin.dvorak@matfyz.cz`
odevzdávejte do odevzdávátka v souboru s příponou `.lean` nebo `.txt`
(prostý text)

Téma 3 – Hex

Vzhledem k tomu, že ke 3. dílu nepřišlo příliš mnoho řešení, rozhodli jsme se prozatím nepřidávat k tématku Hex nový obsah. Pokud byste rádi dále řešili něco matematického, doporučujeme pustit se do dokazování v Leanu. Pokud si chcete dále hrát s jinými nepraktickými věcmi (než matematikou), můžete zkusit náš nový programovací jazyk v tématku FlatFox#.

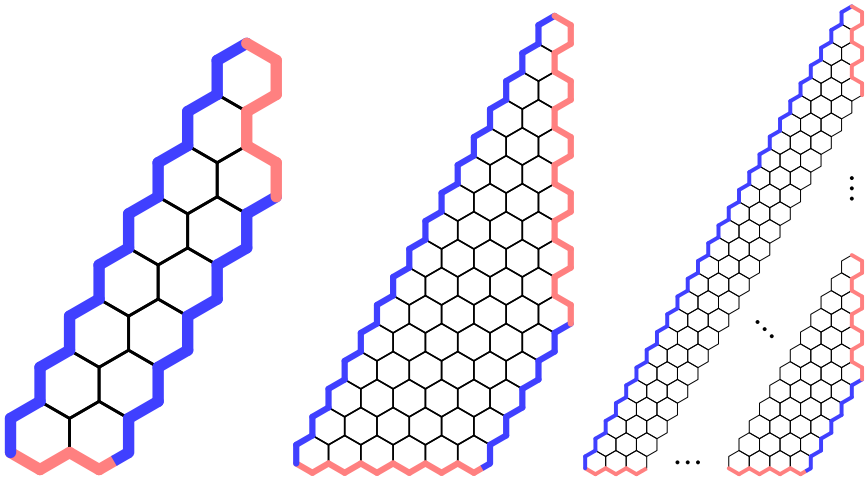
Problém 4.1: *V případě, že byste velmi toužili po řešení tohoto tématka, Hexu, můžete i nadále odevzdávat řešení/články. Kromě již omšelého hledání výherní strategie na různých plochách už můžete zkusit najít něco zajímavého o Hexu na internetu (nezapomeňte uvést zdroje) a hezky to shrnout.*

Případně se můžete začíst do (anglického) článku

`https://arxiv.org/pdf/2201.06475.pdf`,

jímž se toto tématko inspirovalo, a shrnout nám něco o nekonečném Hexu. Pozor, je to opravdu vědecký článek, takže je to opravdu náročné čtení a od kapitoly 5 je bez hlubších znalostí nečitelný. V případě, že vás ani angličtina ani náročnost neodradily, nebojte se ozvat, pokud něčemu z toho nerozumíte (po jazykové nebo odborné stránce), můžeme to probrat a může z toho být hezký článek.

Bětka; `betka.n@centrum.cz`
Jidáš; `jonas.havelka@volny.cz`
odevzdávejte do odevzdávátka

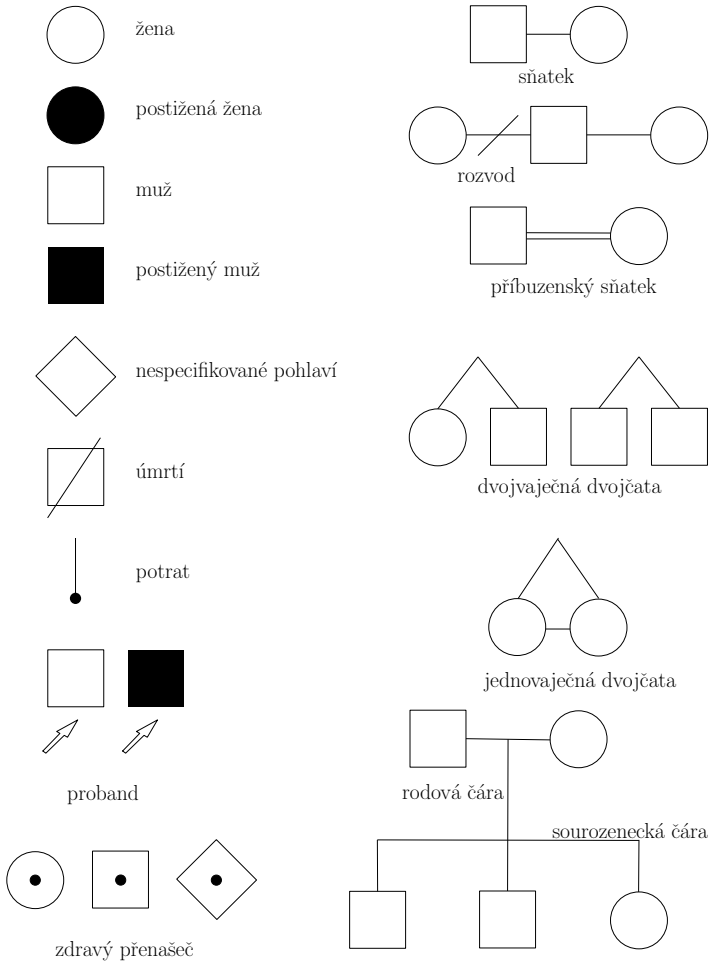


Obrázek 1: Ultimátní otázka: Mějme lichoběžník vzniklý z pravoúhlého trojúhelníku uříznutím vrcholu s pravým úhlem. Nechť jsou ramena lichoběžníku červená a základny modré. Existuje pro každou délku menší základny dostatečná výška lichoběžníku, při které už vyhraje červený?)

Téma 4 – Genetika

V minulém díle jsme si představili základy genetiky. K nim patří i spousta pojmů. Ty doporučujeme mít po ruce i při řešení následujících čísel. Dnes si představíme *genealogii* – nauku o rodokmenu. Poznatky z tohoto oboru se hojně využívají například v medicíně. Určitě se vás někdy lékař zeptal na rodinnou anamnézu, tedy jaké nemoci měli a mají vaši příbuzní. Není to jen tak, spousta problémů je vrozených a právě studium rodiny a rodokmenu dokáže odhalit mnoho informací, které můžou být užitečné např. při volbě léčby.

Z pojmů, které byly vysvětleny v minulém díle, budeme potřebovat především termíny *recesivní* a *dominantní*. Dále si musíme určit pravidla, podle kterých budeme tvořit grafické schéma rodokmenu. V genealogii jsou zavedené jednotné značky, podle kterých se rodokmen zakresluje:



Symbol pro *nespecifikované pohlaví* používáme nejčastěji u plodu, jehož pohlaví zatím neznáme. Může se ale použít pro jakéhokoli jedince s neznámým pohlavím. *Proband* je jedinec, jehož rodokmen zkoumáme. Většinou chceme vědět, jakým způsobem se dědí určitý znak (např. onemocnění), který se u probanda projevil, nebo nás zajímá, s jakou pravděpodobností se u něj daný znak v budoucnu objeví, případně zda ho může přenést na své potomky. Symbol pro *zdravého přenašeče* do rodokmenu většinou doplňujeme až zpětně, když je jisté, že daný jedinec znak přenesl do další generace. Přenašečství se u jedinců nijak neprojevuje, proto ho nejsme schopni vždy zaznamenat. Pokud člen rodokmenu nese symbol pro zdravého jedince, neznamená to, že není přenašeč. Je to spíše doplňkový symbol, který slouží k zaznamenání cesty nemoci napříč již dobře prozkoumaným rodokmenem.

Genetická informace každého živočicha je v podobě DNA rozdělena na segmenty – chromozomy. Člověk má 23 párů chromozomů – jednu sadu s počtem 23 od matky, druhou od otce. Z toho je jeden pár chromozomů pohlavní – nese informaci mimo jiné o pohlaví jedince. Geneticky přenosné nemoci jsou způsobeny mutací (změnou v DNA) a v naprosté většině je lze přiřadit k určitému chromozomu. (Pár nemocím, které se dědí jinak, se budeme věnovat později.) Pokud se změna v DNA způsobující onemocnění nachází na pohlavních chromozomech (gonozomech), označujeme toto onemocnění jako *gonozomální*. Alely na pohlavních chromozomem značíme X/x nebo Y. Pokud se nachází na jednom ze zbylých chromozomů (autozomů), říkáme onemocnění *autozomální*. Alely na autozomech značíme A/a. Velké písmeno značí dominanci, malé recesivitu. Alely způsobující nemoc budeme značit indexem n (např. A_n – dominantní alela na autozomu způsobující nemoc).

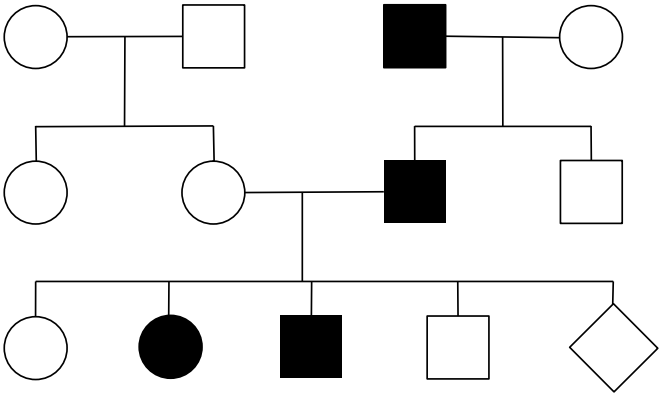
Autozomální nemoci

Autozomálně dominantní

Autozomálně dominantní onemocnění je vázané na autozomální chromozom, jeho výskyt tedy není ovlivněn pohlavím jedince a dědí se stejně u mužů i žen. Zároveň je přenos znaku podmíněn dominantní alelou, fenotyp se tak projeví u každého jedince, který tuto alelu nese – u heterozygotů ($A_n a$) i dominantních homozygotů ($A_n A_n$). Fenotypově zdraví jedinci tedy mutantní alelu určitě nenesou, a nemůžou ji tedy přenést do dalších generací. Zajímavostí u tohoto typu přenosu nemocí je, že dominantní homozygotní forma bývá často letální – neslučitelná s životem. Můžeme si to odůvodnit tím, že jeden chromozom z páru je porouchaný, tedy nese návod pro výrobu defektního produktu - nefunkčního proteinu, který vede k projevům nemoci. U heterozygotů se zároveň podle návodu na druhém zdravém chromozomu vyrábí i menší množství funkčního proteinu, který je potřebný k životu. Částečně se tak vyrovnává deficit způsobený porouchaným chromozomem, i když projevu nemoci to nezabrání. Ale u dominantních homozygotů se žádný funkční protein netvoří, a tak často nemoc vede ke smrti ještě před narozením.

Když se pak díváme na rodokmen, mluvíme o *vertikálním typu dědičnosti*. Ten splňuje tyto podmínky:

1. Pokud je postiženo dítě, alespoň jeden rodič je postižen.
2. Nemoc se vyskytuje prakticky v každé generaci.
3. U postižených není statistická převaha jednoho pohlaví.



Obrázek 2: Příklad rodokmenu autozomálně dominantního onemocnění

Nyní si pro zajímavost ukážeme pár příkladů tohoto typu přenosu nemoci. Jednou z nejznámějších nemocí je *achondroplázie*, což se dá přeložit jako „bez tvorby chrupavky“. Při tomto onemocnění je postižen růst kostí do délky, jedinci tedy mají zkrácené kosti, především kosti končetin. Z toho důvodu se této nemoci říká také dwarfismus, což ovšem může být urážlivé označení. Pacienti také mívají problémy s klouby a často je postihují záněty středního ucha. Intelekt však mají zcela normální. Tato nemoc je způsobena mutací genu FGFR3 na 4. chromozomu a vyskytuje se u zhruba 1 z 15 000 narozených dětí, tedy má incidenci 1:15 000. Další nemocí je například *Huntingtonova choroba*, ta se dostala do povědomí společnosti hlavně díky seriálu Dr. House. O těchto nemocech si můžete zjistit více už na vlastní pěst.

Autozomálně recesivní

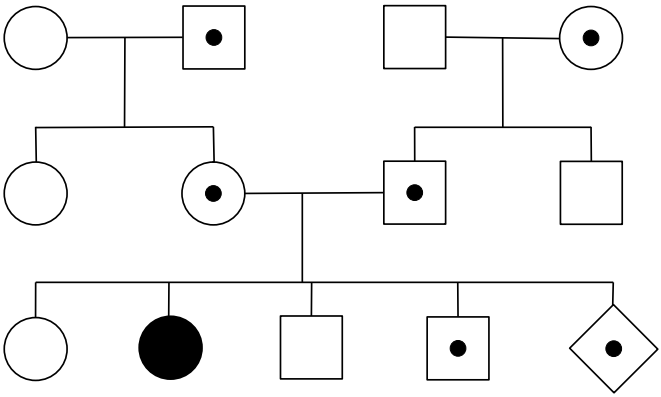
Jak plyne z názvu, u autozomálně recesivních onemocnění je znak vázaný na některý z autozomů a je zastoupen recesivní alelou. Znak se tedy projevuje jen u homozygotů $a_n a_n$, tedy jedinců, kteří mutovanou alelu daného genu zdědili od obou rodičů. Heterozygot (Aa_n) je v tomto případě jen bezpříznakový přenašeč mutované alely.

Na rodokmenu pak vidíme *horizontální typ dědičnosti*, u toho nejčastěji platí:

1. Rodiče postiženého jsou obvykle heterozygoti.

2. Nemoc se projevuje v jedné generaci u sourozenců.

3. Není převaha jednoho pohlaví.



Obrázek 3: Příklad rodokmenu autozomálně recesivního onemocnění

Jednou z nejznámějších takto přenášenou nemocí je *srpkovitá anémie*, o které jsme psali již v minulém díle. Dále se tímto způsobem dědí například *fenylketonurie* či *cystická fibróza*.

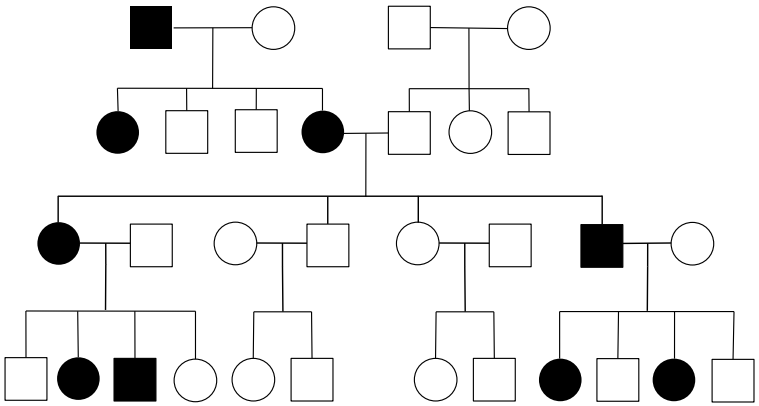
Gonozomální nemoci

Jak už bylo řečeno, jde o nemoci, které jsou vázány na pohlavní chromozom X nebo Y. Ženy nesou chromozomy X a X, muži X a Y – jsou tedy hemizygoti, nesou od každého genu na těchto chromozomech jen jednu kopii. V dědičnosti se to projeví tak, že znaky vázané na gonozom se přenášejí různě často v závislosti na pohlaví jedince.

Gonozomálně dominantní – X vázané

Alela způsobující onemocnění je v tomto případě dominantní a nachází se na chromozomu X. Při tomto způsobu dědičnosti bývají ženy postiženy častěji než muži. Jde o jednoduchou pravděpodobnost – žena může zdědit mutantní formu genu od otce nebo od matky (případně od obou), zatímco muž ji může získat jedině od matky (od otce zdědil chromozom Y). Pokud je nemocný jen otec, budou nemocné i všechny jeho dcery, zatímco všichni synové budou zdraví. Pokud je nemocná jen matka, tak mají syn i dcera stejné šance nemoc zdědit, protože každý od matky zdědí jeden X chromozom.

Tento typ dědičnosti najdeme například u nemoci s názvem *vitamin D – resistentní rachitis*. Vitamín D pomáhá vstřebávání vápníku ze střeva a jeho ukládání do kostí – umožňuje zdravý růst a vývoj kostí. Při nepřítomnosti vitamínu D nejsou kosti dostatečně zásobené vápníkem, jsou slabé a neunesou velké zatížení,



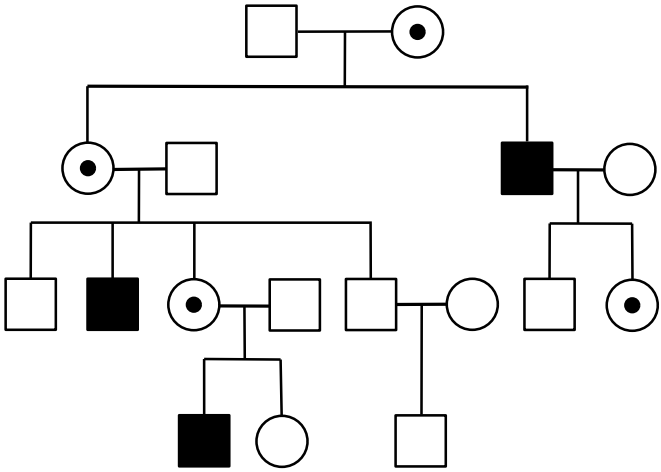
Obrázek 4: Příklad rodokmenu gonozomálně dominantního onemocnění

a proto se deformují. Vitamín D – rezistentní rachitis není způsobena přímo nedostatkem vitamínu D, ale tím, že tělo na vitamín nereaguje, což má stejné příznaky, jako když tam vitamín není. Vápník se do kostí nedostává v dostatečné formě ani když je vitamínu D v těle dostatek.

Gonozomálně recesivní – X vázané

Posledním běžným typem geneticky podmíněných chorob jsou gonozomálně recesivní X vázané nemoci. Ty postihují většinou jen muže. Jelikož jde o recesivní dědičnost, nemoc se projeví jen tehdy, když alela nesoucí onemocnění převládá nad zdravou. Muž má jen jeden chromozom X, takže pokud je na něm mutovaná alela, projeví se u něj nemoc v každém případě ($x_n Y$). Žena má chromozomy X dva, nemoc se tedy projeví jen v případě, že mutovaný gen je na obou chromozómech $X(x_n x_n)$, podobně jako u autozomálně recesivní dědičnosti. Což není moc časté, protože mutovaných recesivních alel je v populaci málo. Pokud má žena mutovaný pouze jeden chromozom X, je jen přenašečkou bez příznaků onemocnění ($x_n X$).

Z nemocí přenášených tímto způsobem určitě stojí za zmínku *„královská nemoc“*, oficiálně známá jako *hemofilie B*. Incidence je asi 1:10 000 mužů. Nemoc narušuje tvorbu srážlivých faktorů, jejichž nedostatek vede k rozvoji poruchy srážlivosti krve. Krev se sráží jen pomalu nebo vůbec, a proto dochází k častým krvácením do kůže, kloubů, krvácení z nosu apod. Krvácení může vyvolat jakékoli nepatrné zranění či bouchnutí, často ani příčinu nenajdeme. Přívlastek „královská nemoc“ dostala hemofilie A proto, že byla pozorována v královské rodině královny Viktorie, která sama byla přenašečkou tohoto onemocnění. Měla hodně dětí, ale nás zajímají pouze dvě dcery přenašečky, nemocný syn a zdravý syn. Zdravý syn Edward VII. zůstal v Británii a stal se králem, jeho rodová větev nemoc již nikdy neměla. Ovšem zbylé dcery a nemocný syn přenesli nemoc do dalších generací. Jedna z dcer se stala předkem ruské a německé královské rodiny, a tak obě tyto



Obrázek 5: Příklad rodokmenu gonozomálně recesivního onemocnění

rodiny na nemoc dále trpěly. Druhá z dcer nemoc přepravila na španělský královský dvůr. Nemocný syn pak nemoc rozšířil do Albánie. O dalších zajímavých nemocech si můžete přečíst sami.

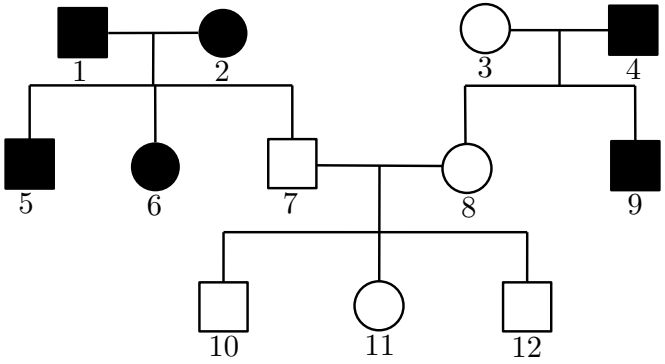
Y vázané

Y vázané znaky jsou způsobené mutací na chromozomu *Y*, dědí se tedy jen po mužské linii. *Y* vázaná dědičnost je vzácná, protože chromozom *Y* nese jen malý počet genů. Dosud není známé žádné onemocnění s tímto typem dědičnosti. Dědí se tak ale například předčasná plešatost.

Mimojaderný typ dědičnosti

Existuje ještě úplně jiný typ dědičnosti, kterému říkáme *mimojaderný*, protože dědičná informace není vázaná na genetickou informaci obsaženou v jádře buněk – tedy jadernou DNA. Malé množství DNA mají živočichové taky v mitochondriích. Mitochondrie jsou semiautonomní organely, což znamená, že ve většině dějů jsou závislé na buňce, ve které se vyskytují, ale pár funkcí dokážou provádět samy, nezávisle na buňce. To jim umožňuje právě vlastní DNA. Samy si vyrábí některé proteiny a dělí se zvlášť. U lidí platí, že zatímco vajíčko je velká buňka, která mitochondrie a další organely obsahuje ve velkém množství, u spermií dochází k redukci velikosti buňky na nutné minimum, které zahrnuje jen malý počet mitochondrií. Navíc je i tento nízký počet zničen po splynutí s vajíčkem. Genetická informace obsažené v mitochondriích se tedy dědí pouze maternálně. Tento typ dědičnosti nazýváme *mitochondriální*. Jako příklady slouží jeden typ slepoty s názvem *Leberova hereditární neuropatie optiku* nebo maternálně dědičný *diabetes mellitus*.

Úloha 4.1 [2b]: *Poznej, jakému typu dědičnosti patří rodokmen na obrázku 6. Odůvodni svoje řešení a uveď genotypy jedinců 1, 6 a 12.*



Obrázek 6: Rodokmen k úloze 4.1

Úloha 4.2 [2b]: *S jakou pravděpodobností se dvěma jedincům s achondroplázií narodí dítě se stejným postižením?*

Úloha 4.3 [2b]: *S jakou pravděpodobností budou trpět hemofilií A děti zdravého muže a ženy přenašečky? Jak se procenta změní u dětí muže hemofilika a zdravé ženy?*

Úloha 4.4 [3b]: *S jakou pravděpodobností budou postiženy děti gonozomálně recesivním X vázaným onemocněním, jestliže:*

1. *matka je heterozygot Xx_n a otec hemizygot $x_n Y$;*
2. *matka je zdravá XX a otec hemizygot $x_n Y$;*
3. *matka je heterozygot Xx_n, otec zdráv XY;*
4. *matka je dominantní homozygot XX, otec zdravý XY;*
5. *matka je dominantní homozygot XX, otec hemizygot $x_n Y$.*

Problém 4.5: *Jaká další gonozomálně recesivní X vázaná nemoc existuje, čím ti přijde zajímavá, jak by vypadal její rodokmen?*

Problém 4.6: *Zkus se zaměřit na jeden výrazný dědičný znak a sledovat, jak se dědí ve vaší rodině nebo v rodině tvých známých. Zakresli váš rodokmen a výskyt znaku napříč příbuznými. Zkus popsat možné genotypy jednotlivých členů a zamysli se, jaký způsob dědičnosti by daný znak mohl mít. Jako znak si vyber něco*

jednoduchého a zjevného (nemusí to být nějaké onemocnění), např. barvu očí, vlasů, kudrnatost vlasů, hudební sluch, výraznou výšku postavy. Něco, o čem se dá jasně říct, zda se to u jednotlivých členů rodiny vyskytuje nebo ne. Pro posouzení typu dědičnosti by bylo vhodné mít aspoň 3 generace, když je neseženeš, snaž se hledat v rodině aspoň do šířky.

Vzorová řešení 2. dílu

Úloha 2.1

Zadání:
Máme rostlinu, její barva se předává úplnou dominancí mendelovsky, kdy fialová barva je dominantní a červená recesivní. Oba potomci rostliny mají fialovou barvu, jaký je možný genotyp a fenotyp jejich rodičů?

Řešení od Doc.^{MM} Jany Uglickich:

Pokud se fialová barva přenáší dominantně, potom bude jedinec fialový, pokud bude mít genotyp AA nebo Aa. Zároveň stačí, aby se v Punnettově čtverci vyskytly alespoň dva potomci s tímto genotypem.

V zásadě existuje $\binom{3}{2} + 3 = 6$ dvojic, které můžu křížit (třikrát jedinci s navzájem různými genotypy a třikrát jedinci se stejnými). Ze zákona o uniformitě hybridů plyne, že křížením rozdílných homozygotů dostanu heterozygoty, a protože se gen přenáší dominantně, budou všichni potomci fialoví. Zároveň jde odvodit, že křížením dvou stejných homozygotů dostanu opět jen ty samé homozygoty (ukazovat Punnettův čtverec v tomto případě považuji za zbytečné). Tím považuji za vyřízené kombinace:

- AA (fialový), aa (červený) – dostanu jen Aa, tedy fialové jedince;
- AA (fialový), AA (fialový) – dostanu jen AA, tedy fialové jedince;
- aa (červený), aa (červený) – dostanu jen aa, tedy červené jedince.

Zbývají tři kombinace, a to AA a Aa, aa a Aa, Aa a Aa:

	A	A
A	AA	AA
a	Aa	Aa

	a	a
A	Aa	Aa
a	aa	aa

	A	a
A	AA	Aa
a	Aa	aa

Ve všech třech případech je možné získat alespoň dva fialové potomky s genotypem AA nebo Aa. Tím pádem, dva fialové potomky je možné získat křížením následujících dvojic:

- AA (fialový), aa (červený);
- AA (fialový), AA (fialový);

- AA (fialový), Aa (fialový);
- aa (červený), Aa (fialový);
- Aa (fialový), Aa (fialový).

Úloha 2.2

Zadání:
S jakou pravděpodobností budou mít potomci určitý genotyp a fenotyp při křížení každých rodičů z předchozího příkladu? Řešení očekáváme podložené Punnettovým čtvercem nebo štěpným poměrem.

Řešení od Doc.^{MM} Jany Uglickich:

1. $AA + aa$:

	A	A
a	Aa	Aa
a	Aa	Aa

Tabulka 1: Genotypový štěpný poměr 4:0, fenotypový štěpný poměr 4:0.

Při křížení jedinců s geny AA a aa vznikne určitě (tj. s pravděpodobností 1) heterozygot, který bude mít fialové listy.

2. $AA + AA$:

	A	A
A	AA	AA
A	AA	AA

Tabulka 2: Genotypový štěpný poměr 4:0, fenotypový štěpný poměr 4:0.

Při křížení dominantních homozygotů vznikne určitě (tj. s pravděpodobností 1) opět dominantní homozygot, který bude mít také fialové listy.

3. $AA + Aa$:

	A	A
A	AA	AA
a	Aa	Aa

Tabulka 3: Genotypový štěpný poměr 1:1, fenotypový štěpný poměr 4:0.

Při křížení dominantního homozygota s heterozygotem vznikne s pravděpodobností 0.5 opět dominantní homozygot, s pravděpodobností 0.5 opět heterozygot. Jakýkoli výsledek takového křížení ale bude mít fialové listy.

4. $aa + Aa$:

	a	a
A	Aa	Aa
a	aa	aa

Tabulka 4: Genotypový štěpný poměr 1:1, fenotypový štěpný poměr 1:1.

Při křížení jedinců s geny aa a Aa vznikne s pravděpodobností 0.5 heterozygot s fialovými listy, se stejnou pravděpodobností vznikne recesivní homozygot s červenými listy.

5. $Aa + Aa$:

	A	a
A	AA	Aa
a	Aa	aa

Tabulka 5: Genotypový štěpný poměr 1:2:1 (AA:Aa:aa), fenotypový štěpný poměr 3:1.

Při křížení dvou heterozygotů vznikne s pravděpodobností 0.25 dominantní homozygot (fialové listy), se stejnou pravděpodobností vznikne recesivní homozygot (červené listy) a s pravděpodobností 0.5 heterozygot s fialovými listy. Tím pádem je zde pravděpodobnost 0.75, že potomek bude mít stejný fenotyp jako jeho rodiče.

Úloha 2.3

Zadání:
Potomek rodiče s modrými a rodiče se žlutými květy má květy zelené, jak je to možné?

Řešení od Doc.MM Jany Uglickich:

Podle mě je to způsobené tím, že gen zodpovědný za barvu květů je v tomto případě neúplně dominantní. Kdyby byl úplně dominantní, bude mít dceřinná rostlina buď modré, nebo žluté květy, což nemá, a kdyby byl kodominantní, potom by se na květu vyskytovaly obě barvy. Zelená sice při míchání barev vznikne smícháním žluté a modré, ale označení „zelený květ“ určitě neznamená, že se na květu vyskytují jen žluté a modré fleky namísto „jednolité“ zelené.

Úloha 2.4

Zadání:
Přisedlé ušní lalůčky jsou na recesivní alele – a. Dolíčky ve tvářích jsou také na recesivní alele – b. Kolik procent potomků bude mít volné ušní lalůčky (opak při-

lehlých) a zároveň dolíčky ve tvářích, jestliže jeden z rodičů je recesivní homozygot pro oba tyto znaky a druhý je pro oba znaky heterozygot?

Řešení od Doc.^{MM} Jany Uglickich:

Jeden rodič má tedy genotyp $aabb$ a ten druhý $AaBb$. Punnettův čtverec pro ně vypadá takto:

	ab	ab	ab	ab
AB	$AaBb$	$AaBb$	$AaBb$	$AaBb$
Ab	$\boldsymbol{Aabb}$	$\boldsymbol{Aabb}$	$\boldsymbol{Aabb}$	$\boldsymbol{Aabb}$
aB	$aaBb$	$aaBb$	$aaBb$	$aaBb$
ab	$aabb$	$aabb$	$aabb$	$aabb$

Tabulka 6: Tučné písmo značí potomky s požadovanými vlastnostmi.

Na jeho základě lze říct, že 25 % potomků bude mít jak volné ušní lalůčky (Aa nebo AA, pokud se tedy gen přenáší úplně dominantně), tak dolíčky ve tvářích (bb).

Problém 2.5

Zadání:
Tento díl alespoň stručně shrnoval historii genetiky. Jaké další chvíle byly významné a jaký má tento obor vůbec význam pro společnost?

Řešení od Mgr.^{MM} Barbory Vosáhlové:

Významným objevem byla například genová terapie, při níž je do genomu pacienta vložena sekvence DNA. Tato sekvence kóduje nějaký chybějící nebo nefungující protein. Sice se tento proces stále ještě nevyužívá v běžné medicíně, jelikož se stále potýká s nedokonalostmi, ale očekává se, že jednou by se tímto způsobem daly léčit geneticky vrozené choroby. Již se ovšem tento proces využil při léčení určitých typů rakoviny.

Význam genetiky pro společnost je obrovský. A to nejen v odvětví medicíny, ale i například v zemědělství, kde jsme schopní pomocí křížení a znalosti pravidel genetiky vypěstovat ideální potraviny. Například jsme schopni vypěstovat melouny mnohem větší a s větším obsahem dužiny než dříve. A ještě k tomu bez pecek!

V medicíně jsme zase díky genetice schopni objevit některé geneticky vrozené nemoci ještě u plodu, protože víme, na který gen se tato informace váže. Proto se občas u dětí, které mají riziko vrozených nemocí (například Huntingtonova choroba), dává přednost umělému oplodnění, kdy jsou doktoři schopni zajistit, aby se postižený gen dále nepředával.

Problém 4.7: *Doplňující otázky k problému 2.5: Jaká jsou úskalí genové terapie? Co brání jejich zavedení do běžné praxe?*

Problém 2.6

Zadání:

Představte si, že jste v kůži Mendela a chcete znovuobjevit tři pravidla. Jak byste postupovali? S jakým živočichem, rostlinou nebo na jakém neživém předmětu byste to ukázali? (Představivosti se meze nekladou, tak se vyhněte hrachu.)

Řešení od Mgr.^{MM} Barbory Vosáhlové:

Pokud bych měla znovuobjevit Mendelova pravidla, zvolila bych si za tímto účelem nějaký organismus s krátkým životním cyklem, který se dokáže rychle rozmnožovat, abychom mohli rychle sledovat zděděné vlastnosti. Samozřejmě by také vše zjednodušilo, kdyby tento organismus byl jednoduchý na chov. A myslím, že by mohlo být mnohem zajímavější sledovat nějaké živočichy než rostliny.

Proto bych nejspíše zvolila nějaký hmyz, který by žil krátkou dobu. Nevýhodou by ovšem bylo, že by se dost možná na hmyzu sledovaly zděděné vlastnosti špatně, jelikož by se s ním hůře manipulovalo. Další možností by byly například myši, jelikož se rychle rozmnožují, nic moc nepotřebují a myslím, že by se u nich jednotlivé znaky sledovaly mnohem jednodušeji.

Nyní bych zvolila konkrétní znak, například barvu srsti. Pokud bych následně zkřížila myš s bílou srstí (aa) a myš s hnědou srstí (AA), všichni potomci by měli hnědou srst, což by potvrzovalo první zákon.

Dále bych pokračovala s křížením a pozorovala poměr barvy srsti potomků. Takže bych například zkřížila dvě hnědosrsté myši z nové generace (Aa). U těchto nových potomků by poměr hnědosrstých k bělosrstým myším měl být přibližně 3:1, přestože fenotyp obou rodičů je hnědá srst. To by ukázalo druhý Mendelův zákon o nezávislém rozdělení alel při tvorbě pohlavních buněk.

Pro zkoumání třetího zákonu bych musela zvolit další znak, který bych pozorovala, například barvu očí. Takže bychom mohli sledovat, jak se kombinace alel pro jeden znak projevuje bez ovlivnění kombinace alel pro druhý znak.

Problém 4.8: *Doplňující otázky k problému 2.6: Pokud zkřížíme myš s genotypem aa a myš s genotypem AA, skutečně dostaneme heterozygotní potomstvo, které nese genotyp Aa a fenotyp A.*

Představme si ale, že parentální genotyp neznáme. Pokud tedy zkřížíme například hnědou myš s bílou myší a veškeré potomstvo bude hnědosrsté, co o takové situaci umíme říct za předpokladu, že se barva srsti opravdu dědí mendelovsky? Jakým postupem bychom mohli zjistit něco víc o genotypu těchto myší? Jakým způsobem bychom mohli získat homozygotní jedince z populace jedinců s neznámým genotypem?

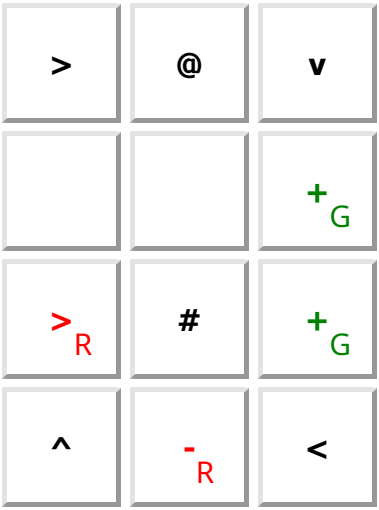
Juli; julie.krimska@mensa.cz
Fofík; martin.fof1@seznam.cz
Ludmila; bujnlu@gmail.com
Jirka; polachj5@gmail.com
odevzdávejte do odevzdávátka

Téma 5 – FlatFox#

Vítejte u představení nového programovacího jazyka. 10 let po vydání světoznámého FlatFox++ (vizte[14] číslo 20.3) založeného na programovacím jazyku FlatFox (vizte[15] číslo 20.1) přicházíme se zcela revolučním jazykem FlatFox# (čteme [flæt foks ša:rp], zkráceně F^{MM}#), který jistě do pár let dnes již zastaralý FlatFox++ nahradí! (A znalci programovacího jazyka C# nebojte, nebude to revoluční tím, že před každý program musíte napsat hromadu[16] `using ...`)

Díl první: Specifikace

O co jde? Mějme (jako na obrázku 7) *les* ve tvaru čtvercové mřížky, po kterém chodí (na čtyři světové strany, tj. nahoru/dolu/doleva/doprava) *liška* začínající směrem doprava ve své *noře* označené `@`.



Obrázek 7: Program pro zdvojnásobení čísla (R, respektive G, v dolním indexu značí červenou, respektive zelenou, pro černobílý tisk)

Taková chytrá liška si pro každou barvu pamatuje nějaké celé číslo (na začátku nulu, pokud není řečeno jinak), těmto číslům budeme také říkat *registry*. Pokud ale potká `+` (nebo `-`) dané barvy, toto číslo zvýší (nebo sníží) o jedna.

Protože je to ale liška líná, tak se pohybuje stále stejným směrem, dokud není nucena směr změnit. K tomu slouží *šipky* (`^`, `>`, `v`, `<`), které mohou být buď

[14] Pro zájemce: `https://mam.mff.cuni.cz/media/cislo/pdf/20/20-3.pdf`, ale není potřeba k řešení tohoto tématka (naopak pro řešení úložek je zcela nežádoucí přečíst si jejich řešení tam).

[15] `https://mam.mff.cuni.cz/media/cislo/pdf/20/20-1.pdf`

[16] `https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using-directive`

bezbarvé (tj. černé) – pak se liška vydá daným směrem vždy – nebo barevné – pak se tím směrem vydá právě tehdy, když si k dané barvě pamatuje nulu (daný registr má hodnotu 0). Takto chodí, dokud nenarazí na *zajímavou úložku*, kterou značíme `#`. V takovém případě se začte do úložky a svůj průchod lesem skončí. V případě dosažení kraje lesa liška přepadává přes Okraj, čímž svůj průchod pochopitelně také končí.

Tak například pokud si liška v lese na obrázku 7 na začátku pamatovala pro červenou (registr R jako red) nějaké kladné[17] číslo, po nalezení úložky si bude pamatovat pro zelenou (registr G jako green) jeho dvojnásobek (bude obíhat pořád dokola, dokud R nebude nula, při každém oběhu sníží R o jedna a G zvýší o dva).

A protože dnes již není programování na papír (ani děrné štítky) v módě, připravili jsme si pro vás na adrese `https://flatfox.moznabude.cz` programovací prostředí. Pozor, program se neukládá, prosíme, ukládejte si ho k sobě průběžně sami (tlačítkem „Uložit“).

Při řešení úloh se snažte o co nejmenší časovou (kolik liška udělá kroků) a prostorovou (kolik si liška musí zapamatovat čísel, tj. kolik použijete registrů) složitost a co nejkratší kód (počet prvků lesa[18]).

Úloha 4.1 [14b]: *Za pomoci `@+-^>v<#` naprogramujte (tj. navrhněte les tak, aby liška skončila na zajímavé úložce a splnila následující; je-li napsáno „nějaké číslo“, „$R = r$“ nebo „$r > 0$“, je cílem napsat jeden program, který danou podúlohu dělá pro všechna taková čísla):*

0.5b Vynulování kladného registru: liška si po opuštění nory (`@`) pamatuje pro červenou nějaké kladné číslo ($R = r > 0$); zařiďte, aby si po dosažení zajímavé úložky (`#`) pamatovala pro červenou nulu ($R = 0$).

1b Přesun registru: liška si po opuštění nory pamatuje pro červenou nějaké kladné číslo ($R = r > 0$); po zastavení si má pamatovat toto číslo v zeleném registru ($G = r$).

1b Kopírování registru[19]: liška si na začátku pamatuje $R = r > 0$ a na konci si chce pamatovat toto číslo v zeleném a modrém registru ($G = B = r$).

1.5b Součet dvou kladných čísel: $R = r > 0$, $G = g > 0$, zařiďte, aby $B = r + g$. (Rozmyslete si i rozdíl.)

1.5b Porovnání dvou kladných čísel: $R = r > 0$, $G = g > 0$, nastavte modrý registr na 1 ($B = 1$), pokud je $r < g$, jinak nastavte modrý na 0 ($B = 0$).

[17] Pokud by bylo toto číslo záporné, tak se liška nikdy nezastaví, takzvaně *se zacyklí*.

[18] Můžete se i zamyslet nad tím, jaký je rozdíl v tom, jestli budeme počítat i prázdná políčka, nebo jen ta neprázdná.

[19] V ukázkové úloze je docela hloupé, že liška během násobení dvěma zapomene původní číslo. To můžeme vyřešit tím, že si ho nejdříve zkopírujeme.

1b Vydělte se zbytkem číslo třemi (a popište, jak by se dělilo větší konstantou): máme $R = r > 0$, chceme G nastavit na celočíselný podíl a B na zbytek po dělení r třemi.

2b Vynásobte dvě kladná čísla.

2.5b Vydělte dvě kladná čísla se zbytkem.

3b Zda je číslo záporné: máme $R = r$, nastavte $G = 1$, pokud $r < 0$, jinak $G = 0$. (Nezapomeňte na to, že se liška musí zastavit.) (V této části úlohy je nejlepší časová složitost docela složitá věc, tedy zde nepožadujeme se jí snažit optimalizovat.)

Problém 4.2: *Dle svého uvážení (budeme bodovat kreativitu) naprogramujte další zajímavé operace.*

Aby program nemusel být obsáhlý, zavedeme základní aritmetiku: pokud liška narazí na barevné *číslo*, zapamatuje si pro tuto barvu toto číslo (zapomene předchozí). Narazí-li na barevný ○, zapamatuje si číslo pro danou barvu navíc i pro černou (tj. nakopíruje daný registr do černého) (černý registr nelze upravovat jiným způsobem). Při potkání barevného ⊕, (⊖, ⊙, ⊘ nebo %), sečte (odečte, vynásobí, celočíselně vydělí, najde zbytek po dělení) obsah černého registru (R) a obsah registru dané barvy a výsledek uloží do dané barvy (černý registr se nezmění). Například: `@`○$_R$⊕$_B$`#` uloží součet R a B do B, `@`○$_R$⊙$_R$`#` uloží druhou mocninu R do R.

Problém 4.3: *Vyberte si nějakou podúlohu z úlohy 4.1 (případně vaši operaci z problému 4.2), která přímo nedělá některou z těchto operací (např. přesun registru, jeho kopírování, nebo zjišťování, zda je číslo záporné) a vyřešte ji snadněji za pomocí nových operací.*

Liška může v lese také narazit na bludný kořen, značený „číslo:číslo“, tj. např. `6:42`. V takovém případě se přesune na pole v konkrétním řádku a sloupci, tj. např. do řádku 6 na pole ve sloupci 42.

Problém 4.4: *K čemu je dobrý bludný kořen? Aneb co je (v programování) funkce, a jak souvisí s bludným kořenem? A proč bychom měli funkce používat?*

Dále si do programu přidáme interakci z vnějším světem: Pokud liška narazí na pohozené barevné `o`, zakřičí na vás číslo, co si pamatuje k dané barvě. Ale protože jak víme, liška nemluví, tak dostaneme (je-li to možné[20]) toto číslo jako jeden ASCII[21] znak. Obdobně při průchodu barevného `i` bude liška naopak poslouchat nás a následující znak vstupu si zapamatuje pod danou barvou. (Po přečtení posledního znaku vstupu bude číst vždy hodnotu 0.)

[20] Není-li číslo v rozsahu ASCII, je chování nedefinované.

[21] Vizte `https://cs.wikipedia.org/wiki/ASCII`, namátkou 48–57 jsou číslice 0–9, 97–122 jsou malá písmenka a–z.

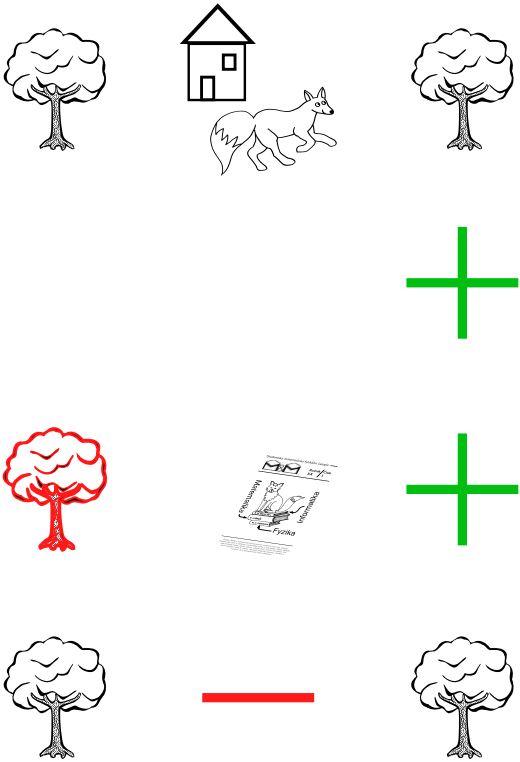
Úloha 4.5 [1b]: *Vypište „Hello world!“.*

Úloha 4.6 [3b]: *Načtěte číslo v desítkové soustavě. Tj. na vstup napíšeme číslo v desítkové soustavě, úkolem je mít v R toto číslo jako hodnotu.*

Problém 4.7: *Naprogramujte další zajímavé programy. Nabízí se například nalezněte n-té (nebo ověřte, zda číslo je): prvočíslo, Fibonacciho číslo, dokonalé číslo, šťastné číslo.*

Problém 4.8: *Co všechno náš programovací jazyk dokáže? Je opravdu třeba všech instrukcí (nestačilo by* `@+-^>v<#`*)?*

Jidáš; `jonas.havelka@volny.cz`
odevzdávejte do odevzdávátka
řešení úložek nejlépe ve formátu .ffs,
tak, jak ho generuje web `https://flatfox.moznabude.cz`



Výsledková listina 2. deadlinu 2. čísla a 1. deadlinu 3. čísla

Poř.	Jméno	R.	$\sum_{-1}$	Témata 1	2	3	4	$\sum_0$	$\sum_1$
1.	Doc.MM J. Uglickich	2	261,0	16,0	26,1			42,1	98,2
2.	Mgr.MM B. Vosáhlová	4	77,2	17,0	6,5			23,5	77,2
3.	Dr.MM L. Zůnová	3	126,8	1,0	8,5			9,5	64,5
4.	Mgr.MM M. Drexlerová	4	60,4	17,5	3,0	4,5	12,5	37,5	60,4
5.	Dr.MM M. Těšitel	3	144,3	2,0	8,0			10,0	58,8
6.	Mgr.MM M. Ambros	1	50,8	6,0	8,8	0,0	10,3	25,1	50,8
7.	Dr.MM O. Nevěřil	2	172,1	18,5	3,1	2,2		23,8	50,2
8.	Mgr.MM L. Votrubová	3	74,3	11,0	2,0		11,1	24,1	49,6
9.	Mgr.MM M. Kadlec	2	55,8	15,0	8,8			23,8	48,3
10.	Dr.MM J. Klementová	2	187,5	19,3	3,0		11,6	33,9	48,1
11.	Doc.MM O. Sedláček	3	213,7		8,5			8,5	47,2
12.	Doc.MM A. Žák	4	200,3	8,5	8,9		7,0	24,4	45,9
13.	Dr.MM R. Novák	4	141,0	16,5	0,2		2,0	18,7	36,8
14.	Bc.MM K. Kučerová	Z8	44,3	3,5	2,0	1,0	10,3	16,8	35,8
15.	Mgr.MM M. Púll	4	85,1						31,5
16.	Bc.MM A. Bakoč	3	30,4		9,0		7,0	16,0	30,4
17.	Dr.MM M. Jarvis	2	179,7						29,9
18.	Mgr.MM M. Urbanová	1	93,7	2,0	1,0	0,7	2,0	5,7	29,4
19.	Bc.MM A. Bihun	4	29,2						29,2
20.	Bc.MM E. Sabol	4	28,7						28,7
21.–22.	Bc.MM M. Stýskala	3	34,0	3,5	5,0			8,5	28,0
	Bc.MM K. Maxera	3	31,0		1,0		7,0	8,0	28,0
23.	Bc.MM T. Pazourek	2	27,5		8,8		4,0	12,8	27,5
24.	Bc.MM J. Koška	4	27,3						27,3
25.	Mgr.MM M. Ulumbekov	2	88,1	4,0	8,0		5,7	17,7	26,4
26.	Bc.MM K. Vomelová	4	31,7						24,5
27.	Mgr.MM K. Plchová	4	85,6	1,5			5,0	6,5	24,2
28.	Mgr.MM A. Stará	3	70,7	0,5			11,0	11,5	23,3
29.	Bc.MM J. Kadlec	3	21,0	8,5				8,5	21,0
30.	Mgr.MM J. Löwenhöffer	3	89,9						20,4
31.	Mgr.MM V. Bartáková	4	80,4				7,0	7,0	20,1
32.	Doc.MM V. Tichý	4	329,8						19,5
33.	V. Sklenár	4	18,5	0,5			5,5	6,0	18,5
34.–35.	O. Hrabě	4	17,5						17,5
	M. Skýpala	3	17,5						17,5
36.	Mgr.MM V. Kučera	2	50,6	4,0	1,0		4,2	9,2	17,0

Poř.	Jméno	R.	$\sum_{-1}$	Témata 1	2	3	4	$\sum_0$	$\sum_1$
37.	Dr.[MM] O. Popovský	4	104,2				5,0	5,0	16,5
38.	Bc.[MM] L. Poljaková	4	41,6						14,7
39.	Dr.[MM] D. Kaňka	2	131,6	9,0				9,0	14,5
40.	K. Česká	3	14,2						14,2
41.–42.	A. Weberová	4	11,6						11,6
	Bc.[MM] M. Jursová	4	43,3	5,5			6,1	11,6	11,6
43.	Dr.[MM] V. Menšíková	2	140,1						11,3
44.	P. Barták	Z9	9,9						9,9
45.–46.	Dr.[MM] J. Tregler	4	116,3						9,4
	Mgr.[MM] M. Rybecký	2	50,6			2,1	5,6	7,7	9,4
47.	A. Stýskala	4	13,9						8,4
48.	M. Ambrosová	Z8	7,8	2,0			5,8	7,8	7,8
49.	A. Migel	1	7,3						7,3
50.	Bc.[MM] V. Vybíral	2	49,2	0,5				0,5	6,8
51.	Bc.[MM] V. Verner	3	48,9						4,0
52.	R. Zelený	3	3,8						3,8
53.–54.	L. Trochová	1	3,0						3,0
	Bc.[MM] R. Petit	3	37,0						3,0
55.	Bc.[MM] K. Menšíková	1	24,9						1,9
56.	S. Teodorovičová	3	8,1						1,0

Sloupeček $\sum_{-1}$ je součet všech bodů získaných v našem semináři, $\sum_0$ je součet bodů v těchto deadlinech a $\sum_1$ součet všech bodů v tomto ročníku. Tituly uvedené v předchozím textu slouží pouze pro účely M&M.

Kontakty:

M&M, OPMK, MFF UK
Ke Karlovu 3
121 16 Praha 2

E-mail: mam@matfyz.cz
Web: mam.matfyz.cz
FB: casopis.MaM

matfyz
M&M