Úvod do Informatiky (IB000 “Úv. Inf.”)

Doc. RNDr. Petr Hliněný, Ph.D.

hlineny@fi.muni.cz

19. prosince 2006

Verze 0.95.

Obsah

1 Základní formalismy: Důkaz a Algoritmus **1**
1.1 Úvod do matematického dokazování 1
1.2 Struktura matematických vět a důkazů 2
1.3 Formální popis algoritmu 4

2 Důkazové techniky, Indukce **5**
2.1 Přehled základních důkazových technik 6
2.2 Věty typu „tehdy a jen tehdy“ 7
2.3 Matematická indukce 8
2.4 Komentáře k matematické indukci 10

3 Množiny, Relace a Funkce **12**
3.1 Pojem množiny 12
3.2 Množinové operace 13
3.3 Porovnávání a určení množin 15
3.4 Relace a funkce mezi (nad) množinami 17
3.5 Posloupnosti a rekurentní vztahy 18

4 Binární relace, Ekvivalence **19**
4.1 Reprezentace konečných relací 20
4.2 Vlastnosti binárních relací 21
4.3 Relace ekvivalence 22
4.4 Rozklady a jejich vztah k ekvivalencím 23

5 Uspořádané množiny, Uzávěry **24**
5.1 Uspořádané množiny 24
5.2 Další pojmy uspořádaných množin 26
5.3 Hasseovské diagramy 27
5.4 Uzávěry relací 28

6 Vlastnosti funkcí a Skládání relací **29**
6.1 Vlastnosti funkcí 29
6.2 Inverzní relace a skládání relací 30
6.3 Skládání relací „v praxi“ 31
6.4 Skládání funkcí, permutace 33
6.5 Induktivní definice množin a funkcí 34

7 Jemný úvod do Logiky **37**
7.1 Výroky v „přirozené“ podobě 37
7.2 (Formální) výroková logika 38
7.3 Jak správně „znegovat formuli“? 39
7.4 Predikátová logika, kvantifikace 41

8 Dokazování vlastností algoritmů **43**
8.1 O „správnosti" programů . . . 43
8.2 Jednoduché indukční dokazování . . . 43
8.3 Algoritmy pro relace . . . 45
8.4 Zajímavé algoritmy aritmetiky . . . 47

9 Jednoduchý deklarativní jazyk **48**
9.1 Popis jednoduchého deklarativního jazyka . . . 49
9.2 Formalizace pojmu „výpočet" . . . 50
9.3 Příklady výpočtů a důkazů . . . 52

10 Důkazové postupy pro algoritmy **54**
10.1 Technika „fixace parametru" . . . 54
10.2 Technika „indukce k součtu parametrů" . . . 54
10.3 Technika „zesílení dokazovaného tvrzení" . . . 56
10.4 Dva „klasické" algoritmy . . . 57

11 Nekonečné množiny a zastavení algoritmu **60**
11.1 O kardinalitě a nekonečných množinách . . . 60
11.2 Algoritmická neřešitelnost problému zastavení . . . 62

12 Délka výpočtu algoritmu **63**
12.1 O významu délky výpočtu algoritmu . . . 64
12.2 Asymptotické značení a odhady funkcí . . . 64

1 Základní formalismy: Důkaz a Algoritmus

Úvod

Začněme nejprve několika obecnými poznámkami ke smyslu a směřování celého našeho předmětu:
Jak sami poznáte, studium informatiky neznamená jen „naučit se nějaký programovací jazyk“, nýbrž zahrnuje celý soubor dalších relevantních předmětů, mezi nimiž najdeme i matematicko–teoretické (formální) základy moderní informatiky. Právě odborný nadhled nad celou informatikou včetně nezbytné formální teorie nejspíše odliší řadového „programátora“, kterých jsou dnes spousty i bez VŠ vzdělání, od skutečného (a mnohem lépe placeného) IT experta.

A na tomto místě přichází náš předmět Úvod do Informatiky, *který (i přes svůj nepříliš obsažný název) vás právě na studium těchto formálních základů moderní informatiky připraví.*

Cíle

Prvním cílem této lekce je formálně rozebrat struktury matematických tvrzení (vět) a jejich formálních důkazů. V druhém sledu se naučíme obdobným způsobem formálně přesně zapisovat algoritmy.

1.1 Úvod do matematického dokazování

Matematika (a tudíž i teoretická informatika jako její součást) se vyznačuje velmi přísnými formálními požadavky na korektnost argumentace. Takto korektně postavená argumentace vede od přijatých předpokladů v elementárních směrem k požadovanému závěru (a nikdy ne naopak!).

Definice 1.1. ***Matematický důkaz*** .
Uvažme matematickou *větu* (neboli tvrzení) tvaru

„Jestliže platí *předpoklady*, pak platí *závěr*“.

Důkaz této věty je konečná posloupnost tvrzení, kde

- každé tvrzení je buď
  - *předpoklad*, nebo
  - obecně přijatá „pravda“ – *axiom*, nebo
  - plyne z předchozích a dříve dokázaných tvrzení podle nějakého „akceptovaného“ logického principu – *odvozovacího pravidla*;
- poslední tvrzení je *závěr*.

`Komentář:` O potřebné úrovni formality matematických důkazů a o běžných důkazových technikách se dozvíme dále v této a příští lekci...
Nyní si prostě celou problematiku uvedeme názornými příklady.

Příklad 1.2. *Uvažujme následující matematické tvrzení (které jistě už znáte).*

Věta. *Jestliže x je součtem dvou lichých čísel, pak x je sudé.*

Poznámka pro připomenutí:

– *Sudé* číslo je celé číslo dělitelné 2, tj. tvaru $2k$.

– *Liché* číslo je celé číslo nedělitelné 2, tj. tvaru $2k+1$.

Důkaz postupuje v následujících formálních krocích:

	tvrzení	zdůvodnění
1)	$a = 2k+1$, k celé	předpoklad
2)	$b = 2l+1$, l celé	předpoklad
3)	$x = a+b = 2k+2l+1+1$	1,2) a komutativita sčítání (axiom)
4)	$x = 2(k+l)+2\cdot 1$	3) a distributivnost násobení
5)	$x = 2(k+l+1)$	4) a opět distributivnost násobení

□

Příklad 1.3. *Dokažte následující tvrzení:*

Věta. *Jestliže x a y jsou racionální čísla pro která platí $x < y$, pak existuje racionální číslo z pro které platí $x < z < y$.*

Důkaz po krocích (s již trochu méně formálním zápisem) zní:

- Nechť $z = \frac{x+y}{2} = x + \frac{y-x}{2} = y - \frac{y-x}{2}$.
- Číslo z je racionální, neboť x a y jsou racionální.
- Platí $z > x$, neboť $\frac{y-x}{2} > 0$.
- Dále platí $z < y$, opět neboť $\frac{y-x}{2} > 0$.
- Celkem $x < z < y$.

□

Poznámka. Alternativní formulace Věty z Příkladu 1.3 mohou znít:

– Pro každé $x, y \in \mathbb{Q}$, kde $x < y$, existuje $z \in \mathbb{Q}$ takové, že $x < z < y$.

– Množina racionálních čísel je hustá.

1.2 Struktura matematických vět a důkazů

První krok formálního důkazu je uvědomit si, *co* se má dokázat, tedy co je *předpoklad* a co *závěr* dokazovaného tvrzení.

Komentář: Příklady formulace matematických *vět*:

∗ Konečná množina má konečně mnoho podmnožin.

∗ $\sin^2(\alpha) + \cos^2(\alpha) = 1$.

∗ Graf je rovinný jestliže neobsahuje podrozdělení K_5 nebo $K_{3,3}$.

Často nám k lepšímu pochopení toho, co je třeba dokázat, pomůže pouhé rozepsání definic pojmů, které se v dané větě vyskytují.

Jak „moc formální“ mají důkazy vlastně být?

- Záleží na tom, komu je důkaz určen — „konzument“ musí být schopen „snadno“ ověřit korektnost každého tvrzení v důkazu a plně pochopit, z čeho vyplývá.
- Je tedy hlavně na vás zvolit tu správnou úroveň formálnosti zápisu vět i důkazů podle situace.

Konstruktivní a existenční důkazy

Z hlediska praktické využitelnosti je potřeba rozlišit tyto dvě kategorie důkazů (třebaže z formálně–matematického pohledu mezi nimi kvalitativní rozdíl není).

- Důkaz Věty 1.3 je *konstruktivní*. Dokázali jsme nejen, že číslo z existuje, ale podali jsme také návod, jak ho pro dané x a y sestrojit.
- *Existenční* důkaz je takový, kde se prokáže existence nějakého objektu bez toho, aby byl podán návod na jeho konstrukci.

Příklad 1.4. *čistě existenčního důkazu.*

Věta. *Existuje program, který vypíše na obrazovku čísla tažená v 45. tahu sportky v roce 2006.*

`Důkaz`: Existuje pouze konečně mnoho možných výsledků losování 45. tahu sportky v roce 2006. Pro každý možný výsledek existuje program, který tento daný výsledek vypíše na obrazovku. Mezi těmito programy je tedy jistě ten, který vypíše právě ten výsledek, který bude v 25. tahu sportky v roce 2006 skutečně vylosován.

□

`Komentář`: To je ale „*podvod*“, že? A přece *není*...
Formálně správně to je prostě tak a tečka.

`Fakt`. Pro informatické i další aplikované disciplíny je důležitá konstruktivnost důkazů vět, které se zde objevují. V matematice ale jsou mnohé příklady užitečných a nenahraditelných existenčních důkazů, třeba tzv. *pravděpodobnostní důkazy*.

Příklad 1.5. *„pravděpodobnostního“ existenčního důkazu.*

Věta. *Na dané množině bodů je zvoleno libovolně n^2 podmnožin, každá o n prvcích. Dokažte pro dostatečně velká n, že všechny body lze obarvit dvěma barvami tak, aby žádná množina nebyla jednobarevná.*

`Důkaz`: U každého bodu si „*hodíme mincí*“ a podle výsledku zvolíme barvu červeně nebo modře. (Nezávislé volby s pravděpodobností $\frac{1}{2}$.) Pravděpodobnost, že zvolených n bodů vyjde jednobarevných, je jen $\frac{2}{2^n} = 2^{1-n}$.

Pro n^2 podmnožin tak je pravděpodobnost, že některá z nich vyjde jednobarevná, shora ohraničená součtem

$$\underbrace{2^{1-n} + \ldots + 2^{1-n}}_{n^2} = \frac{n^2}{2^{n-1}} \to 0\,.$$

Jelikož je toto číslo (pravděpodobnost) pro $n \geq 7$ menší než 1, bude existovat obarvení bez jednobarevných zvolených podmnožin. □

1.3 Formální popis algoritmu

Položme si otázku, co je vlastně algoritmus? Když se na tím zamyslíte, asi zjistíte, že to není tak jednoduché přesně říci. Dokonce je to tolik obtížná otázka, že si zde podáme jen zjednodušenou odpověď.

`Poznámka`: Za definici algoritmu je obecně přijímána tzv. *Church–Turingova teze* tvrdící, že všechny algoritmy lze „simulovat“ na Turingově stroji. Jedná se sice o přesnou, ale značně nepraktickou definici. Mimo Turingova stroje existují i jiné *matematické modely výpočtů*, jako třeba stroj RAM, který je abstrakcí skutečného strojového kódu.

Konvence 1.6. Zjednodušeně zde *algoritmem* rozumíme konečnou posloupnost elementárních (výpočetních) *kroků*, ve které každý další krok využívá vstupní údaje či hodnoty vypočtené v předchozích krocích. Pro zpřehlednění a zkrácení zápisu algoritmu využíváme *řídící konstrukce* – podmíněná větvení a cykly.

> `Komentář:` Vidíte, jak blízké si jsou konstruktivní matematické důkazy a algoritmy (v našem pojetí)?

Příklad 1.7. *Zápis algoritmu pro výpočet průměru z daného pole* `a[]` *s* `n` *prvky.*

Algoritmus.

- Inicializujeme `sum` ← `0` ;
- postupně pro `i=0,1,2,...,n-1` provedeme
  - ∗ `sum` ← `sum+a[i]` ;
- vypíšeme podíl `(sum/n)` . □

Ve „vyšší úrovni“ formálnosti (s jasnějším vyznačením *řídících struktur* algoritmu) se totéž dá zapsat jako:

Algoritmus 1.8. ***Průměr***

z daného pole `a[]` *s* `n` *prvky.*

```
sum ← 0;
for  i ← 0,1,2,...,n-1  do
    sum ← sum+a[i];
done
res ← sum/n;
output res;
```

`Značení`. Pro potřeby symbolického formálního popisu algoritmů v předmětu IB000 si zavedeme následovnou konvenci:

∗ *Proměnné* nebudeme deklarovat ani typovat, pole odlišíme závorkami `p[]`.

∗ *Přiřazení* hodnoty zapisujeme `a` ← `b`, případně `a:=b`, ale nikdy ne `a=b`.

∗ Jako elem. operace je možné použít jakékoliv *aritmetické výrazy* v běžném matematickém zápise. Rozsahem a přesností čísel se zde nezabýváme.

∗ Podmíněné *větvení* uvedeme klíčovými slovy `if ... then ... else ... fi`, kde `else` větev lze vynechat (a někdy, na jednom řádku, i `fi`).

* Pevný *cyklus* uvedeme klíčovými slovy `for ... do ... done`, kde část za `for` musí obsahovat předem danou množinu hodnot pro přiřazování do řídící proměnné.
* *Podmíněný cyklus* uvedeme klíčovými slovy `while ... do ... done`. Zde se může za `while` vyskytovat jakákoliv logická podmínka.
* V zápise používáme jasné odsazování (zleva) podle úrovně zanoření řídících struktur (což jsou `if, for, while`).
* Pokud je to dostatečně jasné, elementární operace nebo podmínky můžeme i ve formálním zápise popsat běžným jazykem.

Malé srovnání závěrem

Jak to tedy je s vhodnou a únosnou mírou formalizace u matematických důkazů i u zápisu algoritmů?

	zcela formální	*běžná úroveň*
`matematika`	formální rozepsání všech elem. kroků (Příklad 1.2)	strukturovaný a matem. přesný text v běžném jazyce
`algoritmy`	rozepsání všech elem. kroků ve výpočetním modelu	strukturovaný rozpis kroků (Algoritmus 1.8), i s využitím běžného jazyka
`programování`	asembler / strojový kód (kde se s ním dnes běžně setkáte?)	„vyšší“ (strukturované) programovací jazyky, například Java

`Komentář:` Pochopitelně se ve všech třech bodech obvykle držíme druhého přístupu, tedy běžné úrovně formality, pokud si specifické podmínky výslovně nevyžadují přístup nejvyšší formality...

Rozšiřující studium

....

2 Důkazové techniky, Indukce

Úvod

Na náš hlubší úvod do matematických formalismů pro informatiku pokračujeme základním přehledem technik matematických důkazů. Z nich pro nás asi nejdůležitější je technika důkazů matematickou indukcí, která je svou podstatou velmi blízká počítačovým programům (jako iterace cyklů).

....

Cíle

Cílem této lekce je popsat základní techniky matematických důkazů, z nichž největší důraz klademe na matematickou indukci. (Pro vysvětlení, z Lekce 1 víme, že matematický *důkaz má „jednotnou formu“ Definice 1.1, ale nyní se věnujeme způsobům, jak takový korektní důkaz vlastně sestavíme.)*

2.1 Přehled základních důkazových technik

V matematice je často používaných několik následujících způsobů – technik, jak k danému tvrzení nalézt korektní formální důkaz. (Uvědomme si, že jedno tvrzení mívá mnoho různých, stejně korektních, důkazů; ty se však mohou výrazně lišit svou složitostí.) Tyto techniky si v bodech shrneme zde:

- *Přímé odvození.* To je způsob, o kterém jsme se dosud bavili.
  Postupujeme přímo od předpokladů k závěru, ale sami poznáte, že taková „přímá“ cesta je obtížně k nalezení.
- *Kontrapozice* (také *obrácením* či *nepřímý důkaz*). Místo věty

  „Jestliže platí *předpoklady*, pak platí *závěr*.“

  budeme dokazovat ekvivalentní větu

  „Jestliže neplatí *závěr*, pak neplatí alespoň jeden z *předpokladů*.“
- *Důkaz sporem*. Místo věty

  „Jestliže platí *předpoklady*, pak platí *závěr*.“

  budeme dokazovat větu

  „Jestliže platí *předpoklady* a platí *opak závěru*, pak platí opak jednoho z *předpokladů* (nebo platí jiné *zjevně nepravdivé tvrzení*).“
- *Matematická indukce.* Pokročilá technika, kterou zde popíšeme později...

Tyto techniky jsou asi nejlépe ilustrovány následovnými příklady důkazů.

Příklad důkazu kontrapozicí

`Definice`: *Prvočíslo* $p > 1$ nemá jiné dělitele než 1 a p.

Příklad 2.1. *na důkaz kontrapozicí (obrácením).*

Věta. *Jestliže p je prvočíslo větší než 2, pak p je liché.*

`Důkaz`: *Obráceného tvrzení* – budeme tedy dokazovat, že je-li p sudé, pak p buď není větší než 2, nebo p není prvočíslo. Jsou dvě možnosti:

- $p \leq 2$. Pak p není větší než 2.
- $p > 2$. Pak $p = 2.k$ pro nějaké celé $k > 1$, tedy p není prvočíslo. □

`Poznámka`: Důkazy kontrapozicí pracují s *negací* (opakem) *předpokladů* a *závěru*. Je-li např. *závěr* komplikované tvrzení tvaru

„z toho, že z A a B plyne C vyplývá, že z A nebo C plyne A a B“,

není pouhou intuicí snadné zjistit, co je vlastně jeho negací. Jak uvidíme v pozdějších lekcích, užitím jednoduché induktivní metody lze podobná tvrzení negovat zcela *mechanicky*.

Příklady důkazu sporem

Příklad 2.2. *Jiný přístup k Důkazu 2.1.*

Věta. *Jestliže p je prvočíslo větší než 2, pak p je liché.*

`Důkaz` *sporem*: Nechť tedy p je prvočíslo větší než 2, které je sudé. Pak $p = 2 \cdot k$ pro nějaké $k > 1$, tedy p není prvočíslo, spor (s předpokladem, že p je prvočíslo). □

Důkaz sporem je natolik specifický a důležitý v matematice, že si zaslouží širší vysvětlení. Co je vlastně jeho podstatou? Je to (zcela přirozený) předpoklad, že v konzistentní teorii nelze zároveň odvodit tvrzení i jeho negaci. Jestliže tedy ve schématu

> „Jestliže platí *předpoklady* a platí *opak závěru*, pak platí opak jednoho z *předpokladů*, nebo platí jiné *zjevně nepravdivé tvrzení*.“

odvodíme k některému předpokladu jeho spor, nebo případně jiné tvrzení, které odporuje všeobecně přijatým faktům (například $0 = 1$), pak něco musí být „špatně“. Co však v našem tvrzení může (nezapomeňte předpoklad konzistence) být chybné? Původní předpoklady byly dány, takže zbývá jedině náš dodatečný předpoklad, že platí *opak závěru*. Tudíž opak závěru nemůže nikdy platit a dvojí negací odvodíme, že platí původní závěr.

Příklad 2.3.

Věta. *Číslo $\sqrt{2}$ není racionální.*

`Důkaz` *sporem*: Nechť tedy $\sqrt{2}$ je racionální, tj. nechť existují nesoudělná celá kladná čísla r, s taková, že $\sqrt{2} = r/s$.

– Pak $2 = r^2/s^2$, tedy $r^2 = 2 \cdot s^2$, proto r^2 je dělitelné dvěma. Z toho plyne, že i r je dělitelné dvěma (proč?).

– Jelikož r je dělitelné dvěma, je r^2 dělitelné dokonce čtyřmi, tedy $r^2 = 4 \cdot m$ pro nějaké m. Pak ale také $4 \cdot m = 2 \cdot s^2$, tedy $2 \cdot m = s^2$ a proto s^2 je dělitelné dvěma.

– Z toho plyne, že s je také dělitelné dvěma. Celkem dostáváme, že r i s jsou dělitelné dvěma, jsou tedy soudělná a to je spor (s definicí racionálního čísla). □

`Komentář:` „Nevíte-li, jak nějakou větu dokázat, zkuste důkaz sporem…“

2.2 Věty typu „tehdy a jen tehdy“

Uvažujme nyní (v matematice poměrně hojné) *věty* tvaru

> „Nechť platí *předpoklady* P. Pak *tvrzení* A platí *tehdy a jen tehdy*, platí-li *tvrzení* B.“

Příklady jiných formulací téže věty jsou:

* Za předpokladů P je tvrzení B *nutnou a postačující* podmínkou pro platnost tvrzení A.
* Za předpokladů P je tvrzení A nutnou a postačující podmínkou pro platnost tvrzení B.
* Nechť platí předpoklady P. Pak tvrzení A platí *právě tehdy* když platí tvrzení B.

`Fakt`: *Důkaz* vět tohoto tvaru má vždy dvě části(!). Je třeba dokázat:

- Jestliže platí předpoklady P a tvrzení A, pak platí tvrzení B.
- Jestliže platí předpoklady P a tvrzení B, pak platí tvrzení A.

2.3 Matematická indukce

Pokud se souhrnně podíváme na důkazové techniky v matematice, všimneme si, že matematická indukce je skoro „dvorní“ důkazovou technikou diskrétní matematiky. To proto, že umožňuje pohodlně dokazovat i složitá tvrzení po jednotlivých (diskrétních) krocích od počátečního.

Uvažme tedy větu ve tvaru:

> *„Pro každé přirozené (celé) $n \geq k_0$ platí $T(n)$.“*

Zde k_0 je nějaké pevné přir. číslo a $T(n)$ je tvrzení parametrizované čís. n. Příkladem je třeba tvrzení:

> *Pro každé $n \geq 0$ platí, že n přímek dělí rovinu*
> *nejvýše na $\frac{1}{2}n(n+1)+1$ oblastí.*

Definice 2.4. ***Princip matematické indukce*** říká, že k důkazu věty

> *„Pro každé přirozené (celé) $n \geq k_0$ platí $T(n)$.“*

stačí ověřit platnost těchto dvou tvrzení:

- $T(k_0)$ (tzv. *báze* neboli základ indukce)
- *Pro každé $n \geq k_0$; jestliže platí $T(n)$,* (*indukční předpoklad*)
  pak platí také $T(n+1)$. (*indukční krok*)

(Formálně řečeno, matematická indukce je axiomem aritmetiky přirozených čísel.)

`Poznámka`: Pozor, v tomto předmětu počítáme 0 za přirozené číslo!

Opět jako v předešlém si tuto techniku ilustrujeme množstvím názorných příkladů.

Příklady důkazů indukcí

Příklad 2.5. *Velmi jednoduchá a přímočará indukce.*

Věta. *Pro každé $n \geq 1$ je stejná pravděpodobnost, že při současném hodu n kostkami bude výsledný součet sudý, jako, že bude lichý.*

`Důkaz`: *Základ indukce* je zde zřejmý: Na jedné kostce (poctivé!) jsou tři lichá a tři sudá čísla, takže obě skupiny padají se stejnou pravděpodobností.

Indukční krok pro $n \geq 1$: Nechť p_n^s pravděpodobnost, že při hodu n kostkami bude výsledný součet sudý, a p_n^l je pravděpodobnost lichého. Podle indukčního předpokladu je $p_n^s = p_n^l = \frac{1}{2}$.

Hoďme navíc $(n+1)$-ní kostkou. Podle toho, zda na ní padne liché nebo sudé číslo, je pravděpodobnost celkového sudého součtu rovna

$$\frac{3}{6}p_n^l + \frac{3}{6}p_n^s = \frac{1}{2}$$

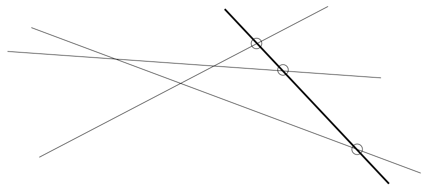
a stejně pro pravděpodobnost celkového lichého součtu. $\square$

Příklad 2.6. *Ukázka důkazové „síly“ principu matematické indukce.*

Věta. *Pro každé $n \geq 0$ platí, že n přímek dělí rovinu nejvýše na*

$$\frac{1}{2}n(n+1)+1$$

oblastí.



`Důkaz`: Pro bázi indukce stačí, že 0 přímek dělí rovinu na jednu část. (Všimněte si také, že 1 přímka dělí rovinu na dvě části, jen pro lepší pochopení důkazu.)

Mějme nyní rovinu rozdělenou n přímkami na nejvýše $\frac{1}{2}n(n+1)+1$ částí. Další, $(n+1)$-ní přímka je rozdělena průsečíky s předchozími přímkami na nejvýše $n+1$ úseků a každý z nich oddělí novou část roviny. Celkem tedy bude rovina rozdělena našimi přímkami na nejvýše tento počet částí:

$$\frac{1}{2}n(n+1)+1 \;+\; (n+1) \;=\; \frac{1}{2}n(n+1)+\frac{1}{2}\cdot 2(n+1) \;+\; 1 \;=\; \frac{1}{2}(n+1)(n+2) \;+\; 1$$

$\square$

Příklad 2.7. *Další indukční důkaz rozepsaný v podrobných krocích.*

Věta. *Pro každé $n \geq 0$ platí $\sum_{j=0}^{n} j = \frac{n(n+1)}{2}$.*

`Důkaz` *indukcí* vzhledem k n.

- *Báze:* Musíme dokázat tvrzení $T(0)$, což je v tomto případě rovnost $\sum_{j=0}^{0} j = \frac{0(0+1)}{2}$. Tato rovnost (zjevně) platí.

- *Indukční krok:* Musíme dokázat následující tvrzení:

Jestliže platí $\sum_{j=0}^{i} j = \frac{i(i+1)}{2}$ kde $i \geq 0$, pak platí $\sum_{j=0}^{i+1} j = \frac{(i+1)(i+2)}{2}$.

Předpokládejme tedy, že $\sum_{j=0}^{i} j = \frac{i(i+1)}{2}$ a pokusme se dokázat, že pak také $\sum_{j=0}^{i+1} j = \frac{(i+1)(i+1+1)}{2} = \frac{(i+1)(i+2)}{2}$. To už plyne úpravou:

$$\sum_{j=0}^{i+1} j \;=\; \sum_{j=0}^{i} j + (i+1) \;=\; \frac{i(i+1)}{2} + (i+1) \;=\; \frac{i(i+1)+2(i+1)}{2} \;=\; \frac{(i+1)(i+2)}{2}$$

Podle principu matematické indukce je celý důkaz hotov. $\square$

2.4 Komentáře k matematické indukci

Pro správné a úspěšné použití indukce v dokazování je vhodné si zapamatovat několik cenných rad:

* Základní trik všech důkazů matematickou indukcí je vhodná reformulace tvrzení $T(n+1)$ tak, aby se „odvolávalo" na tvrzení $T(n)$.
  - Dobře se vždy podívejte, v čem se liší tvrzení $T(n+1)$ od tvrzení $T(n)$. Tento „rozdíl" budete muset v důkaze zdůvodnit.
* Pozor, občas je potřeba „zesílit" tvrzení $T(n)$, aby indukční krok správně „fungoval".
  - Velkým problémem bohužel je, že není možno podat návod, jak vhodné „zesílení" nalézt (ani kdy jej vůbec hledat). Jedná se vlastně o pokusy a „hádání z křišťálové koule".
* Často se chybuje v důkazu indukčního kroku, neboť ten bývá většinou výrazně obtížnější než báze, ale o to zrádnější jsou chyby v samotné zdánlivě snadné bázi!
  - Dejte si dobrý pozor, od které hodnoty $n \geq k_0$ je indukční krok univerzálně platný...

Zesílení indukčního kroku

Příklad 2.8. *Když je nutno indukční krok zesílit...*

Věta. *Pro každé $n \geq 1$ platí*

$$s(n) = \frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \frac{1}{3\cdot 4} + \ldots + \frac{1}{n(n+1)} < 1\,.$$

`Důkaz`: *Báze indukce* je zřejmá, neboť $\frac{1}{1\cdot 2} < 1$.
Co však *indukční krok*? Předpoklad $s(n) < 1$ je sám o sobě „příliš slabý" na to, aby bylo možno tvrdit $s(n+1) = s(n) + \frac{1}{(n+1)(n+2)} < 1$.

Neznamená to ještě, že by tvrzení nebylo platné, jen je potřeba náš indukční předpoklad zesílit. Budeme dokazovat

Tvrzení. *Pro každé přirozené $n \geq 1$ platí $s(n) \leq 1 - \frac{1}{n+1} < 1$.*

To platí pro $n = 1$ (nezapomeňte ověřit!) a dále už úpravou jen dokončíme zesílený indukční krok:

$$s(n+1) \;=\; s(n) + \frac{1}{(n+1)(n+2)} \;\leq\; 1 - \frac{1}{n+1} + \frac{1}{(n+1)(n+2)} =$$

$$= 1 + \frac{-(n+2)+1}{(n+1)(n+2)} = 1 - \frac{1}{n+2}$$

□

Rozšíření báze a předpokladu

Mimo zesilování tvrzení indukčního kroku jsme někdy okolnostmi nuceni i k rozšiřování samotné báze indukce a s ní indukčního předpokladu na více než jednu hodnotu parametru n.

– Můžeme například předpokládat platnost (parametrizovaných) tvrzení $T(n)$ i $T(n+1)$ zároveň, a pak odvozovat platnost $T(n+2)$.

  Toto lze samozřejmě zobecnit na jakýkoliv počet předpokládaných parametrů.

– Můžeme dokonce předpokládat platnost tvrzení $T(j)$ pro všechna $j = k_0, k_0+1, \ldots, n$ najednou a dokazovat $T(n+1)$.

  (Toto typicky využijeme v případech, kdy indukční krok „rozdělí" problém $T(n+1)$ na dvě menší části a z nich pak odvodí platnost $T(n+1)$.)

`Fakt`: Obě prezentovaná „rozšíření" jsou v konečném důsledku jen speciálními instancemi základní matematické indukce; použité rozšířené možnosti pouze zjednodušují formální zápis důkazu.

Příklad 2.9. *Když je nutno rozšířit bázi a indukční předpoklad...*

Věta. *Nechť funkce f pro každé $n \geq 0$ splňuje vztah $f(n+2) = 2f(n+1) - f(n)$. Pokud platí $f(0) = 1$ a zároveň $f(1) = 2$, tak platí $f(n) = n + 1$ pro všechna přirozená $n \geq 0$.*

`Důkaz`: Už samotný pohled na daný vztah $f(n+2) = 2f(n+1) - f(n)$ naznačuje, že bychom měli rozšířit indukční předpoklad (a krok) zhruba takto:

Pro každé $n \geq 0$; jestliže platí $T(n)$; neboli $f(n) = n+1$, a zároveň platí $T(n+1)$; $f(n+1) = n+2$, pak platí také $T(n+2)$; $f(n+2) = n+3$.

Báze indukce – pozor, zde už musíme ověřit dvě hodnoty

$$f(0) = 0 + 1 = 1, \quad f(1) = 1 + 1 = 2\,.$$

Náš *indukční krok* tak nyní může využít celého rozšířeného předpokladu, znalosti hodnot $f(n)$ i $f(n+1)$, pro ověření

$$f(n+2) = 2f(n+1) - f(n) = 2 \cdot (n+1+1) - (n+1) = n+3 = n+2+1\,.$$

□

`Komentář`: Jak by tento důkaz měl být formulován v „tradiční" indukci? („Substitucí" nového tvrzení.)

Závěrem malý „problém"

Příklad 2.10. *Aneb jak snadno lze v matematické indukci udělat chybu.*

Věta. *(„nevěta")*
V každém stádu o $n \geq 1$ koních mají všichni koně stejnou barvu.

`Důkaz` indukcí vzhledem k n.
Báze: Ve stádu o jednom koni mají všichni koně stejnou barvu.
Indukční krok: Nechť $S = \{K_1, \ldots, K_{n+1}\}$ je stádo o $n + 1$ koních. Dokážeme, že všichni koně mají stejnou barvu. Uvažme dvě menší stáda:

– $S' = \{K_1, K_2, \ldots, K_n\}$

– $S'' = \{K_2, \ldots, K_n, K_{n+1}\}$

Podle indukčního předpokladu mají všichni koně ve stádu S' stejnou barvu B'. Podobně všichni koně ve stádu S'' mají podle indukčního předpokladu stejnou barvu B''. Dokážeme, že $B' = B''$, tedy že všichni koně ve stádu S mají stejnou barvu. To ale plyne z toho, že koně $K_2, \dots, K_n$ patří jak do stáda S', tak i do stáda S''. □

`Komentář:` Ale to už je podvod! Vidíte, kde?

Rozšiřující studium

....

3 Množiny, Relace a Funkce

Úvod

V přehledu matematických formalismů informatiky se v této lekci zaměříme na základní „datové typy" matematiky, tj. na množiny, relace a funkce. O množinách jste sice zajisté slyšeli už na základní škole, ale podstatou našeho předmětu je uvést povětšinou neformálně známé pojmy na patřičnou formální úroveň nutnou pro teoretické základy informatiky.

S pojmem funkce jste se také již setkali na nižších stupních škol, ale povětšinou si je asi spojujete jen s aritmetickými a analytickými funkcemi typu $x+1$, x^2-y, či $\sin x$, $1+\cos x^2$, atd. Pojem funkce je však zcela abstraktní a neváže se na žádný analytický vzorec výpočtu.

Cíle

V této lekci si ukážeme první krok k seriózně vybudované matematické teorii množin – tzv. naivní teorii množin, která nám velmi dobře poslouží ve všech konečných případech. Na to navážeme (abstraktní) definicí relace, funkce a posloupnosti, uvedeme si příklady rekurzivně definovaných funkcí a posloupností. Látku relací a funkcí pak dále rozvedeme v následujích dvou lekcích.

3.1 Pojem množiny

Položme si hned na úvod tu nejdůležitější otázku: Co je vlastně ***množina***?
Na tuto otázku bohužel není zcela jednoduchá odpověď... Abychom se vůbec někam v našem úvodu dostali, spokojíme se zatím jen s přirozeným „naivním pohledem".

`Definice` naivní teorie množin: „*Množina je soubor prvků a je svými prvky plně určena.*"

`Komentář:` Pozor, není skutečného rozdílu mezi „množinami" a „prvky". Množiny mohou být prvky jiných množin!

Příklady: $\emptyset$, $\{a, b\}$, $\{b, a\}$, $\{a, b, a\}$, $\{\{a, b\}\}$, $\{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}\}$, $\{x \mid x \text{ je liché přirozené číslo}\}$

`Značení`: Počet prvků (*mohutnost*) množiny A zapisujeme $|A|$.

* $|\emptyset| = 0$, $|\{\emptyset\}| = 1$, $|\{a, b, c\}| = 3$, $|\{\{a, b\}, c\}| = 2$

`Značení` množin a jejich prvků:

* $x \in M$ „x je *prvkem* množiny M".
* některé vlastnosti $a \in \{a, b\}$, $a \notin \{\{a, b\}\}$, $\{a, b\} \in \{\{a, b\}\}$,

* *prázdná* množina $\emptyset$, $\quad a \notin \emptyset$, $\quad \emptyset \in \{\emptyset\}$, $\quad \emptyset \notin \emptyset$,
* *rovnost* množin $\{a,b\} = \{b,a\} = \{a,b,a\}$, $\quad \{a,b\} \neq \{\{a,b\}\}$.

`Definice`: Množina A je *podmnožinou* množiny B, právě když každý prvek A je prvkem B. Píšeme $A \subseteq B$ nebo také $B \supseteq A$; říkáme také, že se jedná o *inkluzi*.

* Platí $\{a\} \subseteq \{a\} \subseteq \{a,b\} \not\subseteq \{\{a,b\}\}$, $\quad \emptyset \subseteq \{\emptyset\}$,
* $A \subset B$ právě když $A \subseteq B$ a $A \neq B$ $\quad$ (A je *vlastní* podmnožinou B).

`Definice`: Dvě množiny jsou si *rovny* $A = B$ právě když $A \subseteq B$ a $B \subseteq A$.

* Podle definice jsou množiny A a B stejné, mají-li stejné prvky.
* Důkaz rovnosti množin $A = B$ má obvykle dvě části: Odděleně se dokáží inkluze $A \subseteq B$ a $B \subseteq A$.

`Značení`: Některé běžné množiny v matematice se značí

* $\mathbb{N} = \{0,1,2,3,\ldots\}$ je množina přirozených čísel,
* $\mathbb{Z} = \{\ldots,-2,-1,0,1,2,3,\ldots\}$ je množina celých čísel,
* $\mathbb{Z}^+ = \{1,2,3,\ldots\}$ je množina celých kladných čísel,
* $\mathbb{Q}$ je množina racionálních čísel (zlomků).
* $\mathbb{R}$ je množina reálných čísel.

`Poznámka`: Tyto uvedené číselné množiny jsou vesměs *nekonečné*, na rozdíl od konečných množin uvažovaných v předchozím „naivním“ pohledu. Pojem nekonečné množiny se přímo v matematice objevil až teprve v 19. století a bylo s ním spojeno několik paradoxů ukazujících, že naivní pohled na teorii množin pro nekonečné množiny nedostačuje. My se k problematice nekonečných množin, Kantorově větě a Russelově paradoxu vrátíme v závěru našeho předmětu.

3.2 Množinové operace

Začněme se základními operacemi, které zajisté na intuitivní úrovni již ovládáte.

`Definice`: *Sjednocení* $\cup$ a *průnik* $\cap$ dvou množin A, B definujeme

$$\begin{aligned} A \cup B &= \{x \mid x \in A \text{ nebo } x \in B\}\,, \\ A \cap B &= \{x \mid x \in A \text{ a současně } x \in B\}\,. \end{aligned}$$

* Příklady $\{a,b,c\} \cup \{a,d\} = \{a,b,c,d\}$, $\quad \{a,b,c\} \cap \{a,d\} = \{a\}$.
* Vždy platí „distributivita“ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ a $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

`Definice`: Pro libovolný počet množin indexovaných pomocí I rozšířeně definujeme

$$\begin{aligned} \textstyle\bigcup_{i\in I} A_i &= \{x \mid x \in A_i \text{ pro nějaké } i \in I\}\,, \\ \textstyle\bigcap_{i\in I} A_i &= \{x \mid x \in A_i \text{ pro každé } i \in I\}\,. \end{aligned}$$

Komentář: Nechť $A_i = \{2.i\}$ pro každé $i \in \mathbb{N}$. Pak $\bigcup_{i\in\mathbb{N}} A_i$ je množina všech sudých přirozených čísel. Nechť $B_i = \{x \mid x \in \mathbb{N}, x \geq i\}$ pro každé $i \in \mathbb{N}$. Pak $\bigcap_{i\in\mathbb{N}} B_i = \emptyset$.

Definice: *Rozdíl* $\setminus$ a *symetrický rozdíl* Δ dvou množin A, B definujeme

$$\begin{aligned} A \setminus B &= \{x \mid x \in A \text{ a současně } x \notin B\}, \\ A \Delta B &= (A \setminus B) \cup (B \setminus A). \end{aligned}$$

* Příklady $\{a,b,c\} \setminus \{a,b,d\} = \{c\}$, $\{a,b,c\}\Delta\{a,b,d\} = \{c,d\}$.
* Vždy platí například $A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$ apod.

Definice: Nechť $A \subseteq M$. *Doplněk A vzhledem k M* je množina $\overline{A} = M \setminus A$.

Komentář: Jedná se o poněkud specifickou operaci, která musí být vztažena vzhledem k *nosné množině M*!

* Je-li $M = \{a,b,c\}$, pak $\overline{\{a,b\}} = \{c\}$. Je-li $M = \{a,b\}$, pak $\overline{\{a,b\}} = \emptyset$.
* Vždy pro $A \subseteq M$ platí $\overline{\overline{A}} = A$ („dvojí“ doplněk).
* Vždy pro $A, B \subseteq M$ platí $\overline{A \cup B} = \overline{A} \cap \overline{B}$ a $\overline{A \cap B} = \overline{A} \cup \overline{B}$. (Viz Vennovy diagramy.)

Uspořádané dvojice a kartézský součin

Zatímco prvky v množinách jsou zcela neuspořádané, jsou mnohé situace, kdy musíme pracovat se „seřazenými“ výčty prvků. V teorii množin lze takovéto seřazení definovat oklikou, například následovně:

Definice: *Uspořádaná dvojice* (a,b) je zadána množinou $\{\{a\},\{a,b\}\}$.

Fakt: Platí $(a,b) = (c,d)$ právě když $a = c$ a současně $b = d$.

Příklad 3.1. *Co je podle definice (a,a)?*
$(a,a) = \{\{a\},\{a,a\}\} = \{\{a\},\{a\}\} = \{\{a\}\}$. □

Definice 3.2. ***Kartézský součin*** dvou množin A, B definujeme jako množinu všech uspořádaných dvojic ze složek z A a B

$$A \times B = \{(a,b) \mid a \in A, b \in B\}.$$

* Příklady $\{a,b\} \times \{a\} = \{(a,a),(b,a)\}$, $\{c\} \times \{a,b\} = \{(c,a),(c,b)\}$.
* Platí $\emptyset \times X = \emptyset$ pro každou množinu X.
* Mnemotechnická pomůcka
$$|A \times B| = |A| \cdot |B|.$$

Definice: Pro každé $k \in \mathbb{N}, k > 0$ definujeme *uspořádanou k-tici* $(a_1, \cdots, a_k)$ induktivně takto

- $(a_1) = a_1$,
- $(a_1, \cdots, a_i, a_{i+1}) = ((a_1, \cdots, a_i), a_{i+1})$.

`Fakt`: Platí $(a_1, \cdots, a_k) = (b_1, \cdots, b_k)$ právě když $a_i = b_i$ pro každé $i \in \mathbb{N}$ kde $1 \leq i \leq k$.

`Definice` *kartézského součinu* více množin: Pro každé $k \in \mathbb{N}$ definujeme

$$A_1 \times \cdots \times A_k = \{(a_1, \cdots, a_k) \mid a_i \in A_i \text{ pro každé } 1 \leq i \leq k\} .$$

* Příklad $\mathbb{Z}^3 = \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} = \{(i, j, k) \mid i, j, k \in \mathbb{Z}\}$.
* Co je A^0? $\{\emptyset\}$, neboť jediná uspořádaná 0-tice je právě prázdná $\emptyset$.

`Poznámka`: Podle uvedené definice není součin asociativní, tj. obecně nemusí platit, že $A \times (B \times C) = (A \times B) \times C$.

V matematické praxi je někdy výhodnější uvažovat „upravenou“ definici, podle níž součin asociativní je. Pro účely této přednášky není podstatné, k jaké definici se přikloníme. Prezentované definice a věty „fungují“ pro obě varianty.

Potenční množina

Definice 3.3. ***Potenční množina*** množiny A,
neboli množina všech podmnožin, je definovaná vztahem

$$2^A = \{B \mid B \subseteq A\} .$$

* Platí například $2^{\{a,b\}} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$,
* $2^\emptyset = \{\emptyset\}$, $2^{\{\emptyset,\{\emptyset\}\}} = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\emptyset, \{\emptyset\}\}\}$,
* $2^{\{a\}\times\{a,b\}} = \{\emptyset, \{(a, a)\}, \{(a, b)\}, \{(a, a), (a, b)\}\}$.

Věta 3.4. *Počet prvků potenční množiny splňuje* $|2^A| = 2^{|A|}$.

`Důkaz`: Stručně indukcí podle $|A|$: Pro $A = \emptyset$ platí $|2^A| = |\{\emptyset\}| = 1$.

Pro každý další prvek $b \notin A$ rozdělíme všechny podmnožiny $A \cup \{b\}$ napolovic na ty neobsahující b a na ty obsahující b, tudíž

$$|2^{A\cup\{b\}}| = 2 \cdot |2^A| = 2^{|A|+1} .$$

□

3.3 Porovnávání a určení množin

Pro obecnou ilustraci formálního množinového kalkulu si ukažme dva důkazy rovností mezi množinovými výrazy. Podobně lze (rozepsáním příslušných definic) rutinně dokazovat další množinové vztahy.

Věta 3.5. *Pro každé dvě množiny* $A, B \subseteq M$ *platí* $\overline{A \cup B} = \overline{A} \cap \overline{B}$.

`Důkaz` v obou směrech rovnosti.

- $\overline{A \cup B} \subseteq \overline{A} \cap \overline{B}$: Pro $x \in M$ platí $x \in \overline{A \cup B}$, právě když $x \notin A \cup B$, neboli když zároveň $x \notin A$ a $x \notin B$. To znamená $x \in \overline{A}$ a zároveň $x \in \overline{B}$, z čehož vyplývá požadované $x \in \overline{A} \cap \overline{B}$.
- $\overline{A \cup B} \supseteq \overline{A} \cap \overline{B}$: Pro $x \in M$ platí $x \in \overline{A} \cap \overline{B}$, právě když $x \in \overline{A}$ a zároveň $x \in \overline{B}$, neboli když zároveň $x \notin A$ a $x \notin B$. To znamená $x \notin A \cup B$, z čehož vyplývá požadované $x \in \overline{A \cup B}$.

□

Věta 3.6. *Pro každé tři množiny A, B, C platí*

$$A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C).$$

`Důkaz.`

- $A \setminus (B \cap C) \subseteq (A \setminus B) \cup (A \setminus C)$: Je-li $x \in A \setminus (B \cap C)$, pak $x \in A$ a zároveň $x \notin (B \cap C)$, neboli $x \notin B$ nebo $x \notin C$. Pro první možnost máme $x \in (A \setminus B)$, pro druhou $x \in (A \setminus C)$.
- Naopak $A \setminus (B \cap C) \supseteq (A \setminus B) \cup (A \setminus C)$: Je-li $x \in (A \setminus B) \cup (A \setminus C)$, pak $x \in (A \setminus B)$ nebo $x \in (A \setminus C)$. Pro první možnost máme $x \in A$ a zároveň $x \notin B$, z čehož plyne $x \in A$ a zároveň $x \notin (B \cap C)$, a tudíž $x \in A \setminus (B \cap C)$. Druhá možnost je analogická. □

Charakteristický vektor (pod)množiny

V případech, kdy všechny uvažované množiny jsou podmnožinami nějaké *nosné množiny X*, což není neobvyklé v programátorských aplikacích, s výhodou využijeme následující reprezentaci množin.

`Definice`: Mějme nosnou množinu $X = \{x_1, x_2, \ldots, x_n\}$. Pro $A \subseteq X$ definujeme charakteristický vektor χ_A jako

$$\chi_A = (c_1, c_2, \ldots, c_n), \text{ kde } c_i = 1 \text{ pro } x_i \in A \text{ a } c_i = 0 \text{ jinak.}$$

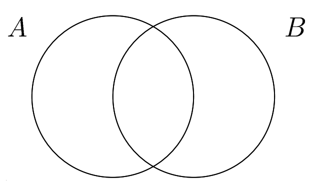
Užitečnost této reprezentace je ilustrována také následovnými fakty.

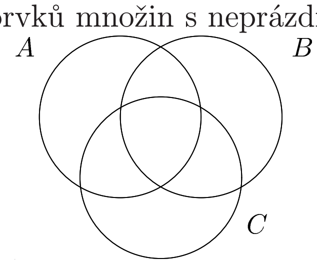
- Platí $A = B$ právě když $\chi_A = \chi_B$.
- Množinové operace jsou realizovány „bitovými funkcemi“

  sjednocení $\sim$ OR, průnik $\sim$ AND, symetrický rozdíl $\sim$ XOR.

Princip inkluze a exkluze

Tento důležitý a zajímavý kombinatorický princip je někdy také nazýván „princip *zapojení a vypojení*“. Používá se ke zjišťování počtů prvků množin s neprázdnými průniky.





Věta 3.7. *Počet prvků ve sjednocení dvou či tří množin spočítáme:*

$$|A \cup B| = |A| + |B| - |A \cap B|$$

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

Komentář: Všimněte si, že větu lze stejně tak využít k výpočtu počtu prvků v průniku množin...

Příklad 3.8. *Z* 1000 *televizí jich při první kontrole na výrobní lince má* 5 *vadnou obrazovku,* 10 *je poškrábaných a* 12 *má jinou závadu. Přitom* 3 *televize mají současně všechny tři vady a* 4 *jiné jsou poškrábané a mají jinou vadu.*
Kolik televizí je celkem vadných?

Řešení: Dosazením do Věty 3.7 zjistíme výsledek 17. □

Poznámka. Jen stručně, bez důkazu a bližšího vysvětlení, si uvedeme obecnou formu principu inkluze a exkluze:

$$\left|\bigcup_{j=1}^{n} A_j\right| = \sum_{\emptyset \neq I \subseteq \{1,\dots,n\}} (-1)^{|I|-1} \cdot \left|\bigcap_{i \in I} A_i\right|$$

(Jeho znalost nebude v předmětu vyžadována.)

3.4 Relace a funkce mezi (nad) množinami

Dalším důležitým základním „datovým typem“ matematiky jsou relace, kterým vzhledem k jejich rozsáhlému použití v informatice věnujeme zvýšenou pozornost. (Hlavní látka o relacích přijde až v následujících lekcích.)

Definice 3.9. ***Relace*** mezi množinami $A_1, \cdots, A_k$, pro $k \in \mathbb{N}$,
je libovolná podmnožina kartézského součinu

$$R \subseteq A_1 \times \cdots \times A_k\,.$$

Pokud $A_1 = \cdots = A_k = A$, hovoříme o *k-ární relaci R na A*.

Komentář: Příklady relací.

* $\{(1, a), (2, a)\}$ je relace mezi $\{1, 2, 3\}$ a $\{a, b\}$.
* $\{(i, 2.i) \mid i \in \mathbb{N}\}$ je binární relace na $\mathbb{N}$.
* $\{(i, j, i + j) \mid i, j \in \mathbb{N}\}$ je ternární relace na $\mathbb{N}$.
* $\{3.i \mid i \in \mathbb{N}\}$ je unární relace na $\mathbb{N}$.
* Jaký význam vlastně mají unární a nulární relace na A?

Uvědomme si, jak obecně je relace definována – její definice umožňuje podchytit skutečně libovolné „vztahy“ mezi prvky téže i různých množin. V praxi se relace velmi široce využívají třeba v relačních databázích...

Funkce mezi množinami

Tradiční „školní“ pojetí pojmu *funkce* jej obvykle identifikuje s nějakým výpočetním (analytickým) předpisem či vzorcem. Pojem funkce je však daleko obecnější a zcela abstraktní.

Definice 3.10. ***(Totální) funkce*** z množiny A do množiny B je relace f mezi A a B taková, že pro každé $x \in A$ existuje právě jedno $y \in B$ takové, že $(x, y) \in f$.

Množina A se nazývá *definiční obor* a množina B *obor hodnot* funkce f.

`Komentář:` Neformálně řečeno, ve funkci f je každé „vstupní“ hodnotě x přiřazena jednoznačně „výstupní“ hodnota y. (V obecné relaci počty „přiřazených“ dvojic neomezujeme...)

`Značení`: Místo $(x, y) \in f$ píšeme obvykle $f(x) = y$.
Zápis $f : A \to B$ říká, že f je funkce s definičním oborem A a oborem hodnot B.
Funkcím se také říká *zobrazení*.

`Komentář:` Příklady funkcí.

* Definujeme funkci $f : \mathbb{N} \to \mathbb{N}$ předpisem $f(x) = x + 8$. Tj. $f = \{(x, x + 8) \mid x \in \mathbb{N}\}$.
* Definujeme funkci *plus* : $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ předpisem $plus(i, j) = i + j$. Tj. $plus = \{(i, j, i + j) \mid i, j \in \mathbb{N}\}$.

`Definice`: Pokud naší definici funkce upravíme tak, že požadujeme pro každé $x \in A$ nejvýše jedno $y \in B$ takové, že $(x, y) \in f$, obdržíme definici *parciální funkce* z A do B.

`Komentář:` V parciální funkci p nemusí být pro některé „vstupní“ hodnoty x funkční hodnota definována.

Pro *nedefinovanou* hodnotu používáme znak $\perp$.

`Komentář:` Další příklady funkcí.

* Definujeme parciální funkci $f : \mathbb{Z} \to \mathbb{N}$ předpisem

$$f(x) = \begin{cases} 3 + x & \text{jestliže } x \geq 0, \\ \perp & \text{jinak.} \end{cases}$$

  Tj. $f = \{(x, 3 + x) \mid x \in \mathbb{N}\}$.

* Také funkce $f : \mathbb{R} \to \mathbb{R}$ daná běžným analytickým předpisem

$$f(x) = \sqrt{x}$$

  je jen parciální – není definována pro $x < 0$.

* Co je relace, přiřazující lidem v ČR jejich rodná čísla?

3.5 Posloupnosti a rekurentní vztahy

Specifickým případem funkcí jsou ty, jež jsou definovány z přirozených čísel – u nich totiž funkční hodnoty můžeme snadno jednu po druhé vypsat jako $f(0), f(1), f(2), \ldots$ a také takto jednoduše je zadat.

`Definice`: Funkce $p : \mathbb{N} \to \mathbb{R}$ se nazývá *posloupnost*. Mimo „funkčního“ zápisu $p(n)$ často používáme „indexovou“ formu zápisu funkční hodnoty p_n.

`Poznámka`: Obor hodnot posloupnosti může být i jiný než reálná čísla. Na posloupnost se také díváme jako na „seřazení“ vybraných prvků z oboru hodnot, s povoleným opakováním hodnot (nemusí být prostá). Také definiční obor posloupnosti může být od nuly nebo i od jedničky, jak je v aplikacích potřeba.

- Příklady posloupností:
  - $*$ $p_0 = 0,\ p_1 = 2, \ldots, p_i = 2i, \ldots$ je posloupnost sudých nezáporných čísel.
  - $*$ $3,\ 3.1,\ 3.14,\ 3.141, \ldots$ je posloupnost postupných dekadických rozvojů π.
  - $*$ $1,\ -1,\ 1,\ -1, \ldots$ je posloupnost určená vztahem $p_i = (-1)^i$, $i \geq 0$.
  - $*$ Pokud chceme stejnou posloupnost $1,\ -1,\ 1,\ -1, \ldots$ zadat jako q_i, $i \geq 1$, tak ji určíme vzorcem $q_i = (-1)^{i-1}$.
- Posloupnost je *rostoucí* či *klesající*, pokud $p_{n+1} > p_n$ či $p_{n+1} < p_n$ pro všechna n.

Rekurentní definice posloupnosti

Slovem *rekurentní* označujeme takové definice (či popisy), které se v jistých bodech odvolávají samy na sebe. (Už jste se setkali s „rekurzí“ při programování? A víte, co znamená?)

Místo nepřehledných formálních definic si *rekurentní vztahy* uvedeme několika názornými ukázkami.

- Zadáme-li posloupnost p_n vztahy $p_0 = 1$ a $p_n = 2p_{n-1}$ pro $n > 0$, pak platí $p_n = 2^n$ pro všechna n.
- Obdobně můžeme zadat posloupnost q_n vztahy $q_1 = 1$ a $q_n = q_{n-1} + n$ pro $n > 1$. Potom platí $q_n = \frac{1}{2}n(n+1)$ pro všechna n. Uměli byste toto dokázat indukcí? Viz Příklad 2.7.
- Známá Fibonacciho posloupnost je zadaná vztahy $f_1 = f_2 = 1$ a $f_n = f_{n-1} + f_{n-2}$ pro $n > 2$.

Rozšiřující studium

....

4 Binární relace, Ekvivalence

Úvod

V návaznosti na předchozí lekci si podrobně rozebereme matematické formalismy relací. Na rozdíl od množin, které se v jisté velmi naivní formě objevují už na základní škole, se relacím v jejich abstraktní podobě moc pozornosti nevěnuje. Avšak na pojem relace velmi brzo narazí (snad) každý informatik při studiu relačních databází.

Není to však jen oblast databází, ale i jiná místa informatiky, kde se relace skrývají či přímo explicitně objevují. Nejčastěji se takto setkáme s binárními relacemi, například vždy, když rozdělujeme objekty podle „shodných“ znaků (relace ekvivalence), nebo když objekty mezi sebou „srovnáváme“ (relace uspořádání). Na tyto dvě základní oblasti se dále zaměříme.

Cíle

Úkolem této lekce je vybudovat matematickou teorii (konečných) relací, s primárním zaměřením na binární relace jako ekvivalence a uspořádání. Ve vztahu k binárním relacím je zavedeno množství pojmů, které jsou později užitečné v různých oblastech matematiky i informatiky. Látka pak plynule pokračuje Lekcí 5.

4.1 Reprezentace konečných relací

Oblast, kde informatici nejčastěji potkají relace, je bezesporu ukládání dat. (Neboť shromažďovaná data, stejně jako relace, především sledují vztahy mezi danými objekty. . .)

Příklad 4.1. *Tabulky relační databáze.*

Definujme následující množiny („elementární typy")

* ZNAK $= \{a, \cdots, z, A, \cdots, Z, mezera\}$,
* CISLICE $= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Dále definujeme tyto množiny („odvozené typy")

* JMENO = ZNAK^{15}, PRIJMENI = ZNAK^{20},
* VEK = CISLICE^{3},
* ZAMESTNANEC = JMENO $\times$ PRIJMENI $\times$ VEK.

Relaci „typu" *ZAMESTNANEC* pak lze reprezentovat tabulkou:

JMENO	*PRIJMENI*	*VEK*
Jan	Novák	42
Petr	Vichr	28
Pavel	Zíma	26

□

`Definice`: *Relační databáze* je konečná množina tabulek. *Schéma databáze* je (zjednodušeně řečeno) množina „typů" jednotlivých tabulek.

Reprezentace binárních relací na množině

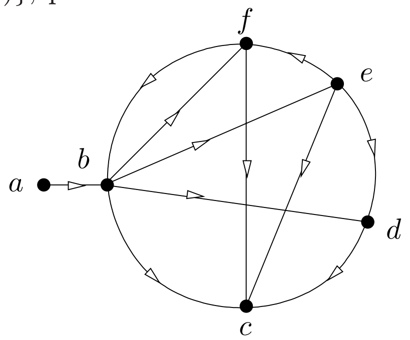
Jistě čtenáři uznají, že zadání relace výčtem jejích složek není pro člověka (na rozdíl od počítače) tím nejpříjemnějším způsobem. Je tedy přirozené se ptát, jak co nejnázorněji takovou relaci, alespoň v její nejčastější binární podobě, ukázat.

`Značení`: Binární relaci $R \subseteq M \times M$ lze jednoznačně znázornit jejím *grafem*.

- Prvky M znázorníme jako body v rovině.
- Prvek $(a, b) \in R$ znázorníme jako *orientovanou hranu* („šipku") z a do b. Je-li $a = b$, pak je touto hranou „smyčka" na a.

`Komentář`: Pozor, nejedná se o „grafy funkcí" známé z analýzy.

Například mějme $M = \{a, b, c, d, e, f\}$ a $R = \{(a, b), (b, c), (b, d), (b, e), (b, f), (d, c), (e, c), (f, c), (e, d), (e, f), (f, b)\}$, pak:



V případě, že R je nekonečná nebo „velká“, může být reprezentace R jejím grafem nepraktická (záleží pak na míře „pravidelnosti“ R).

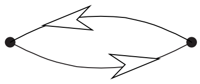
4.2 Vlastnosti binárních relací

Definice 4.2. Nechť $R \subseteq M \times M$. Binární relace R je

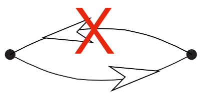
- *reflexivní*, právě když pro každé $a \in M$ platí $(a, a) \in R$;

- *ireflexivní*, právě když pro každé $a \in M$ platí $(a, a) \notin R$;

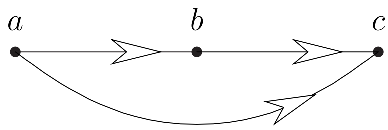
- *symetrická*, právě když pro každé $a, b \in M$ platí, že jestliže $(a, b) \in R$, pak také $(b, a) \in R$;



- *antisymetrická*, právě když pro každé $a, b \in M$ platí, že jestliže $(a, b), (b, a) \in R$, pak $a = b$;



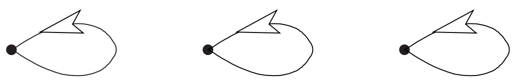
- *tranzitivní*, právě když pro každé $a, b, c \in M$ platí, že jestliže $(a, b), (b, c) \in R$, pak také $(a, c) \in R$.



Následují dva základní typy binárních relací, R je

- relace *ekvivalence*, právě když je R reflexivní, symetrická a tranzitivní;
- *částečné uspořádání*, právě když je R reflexivní, antisymetrická a tranzitivní (často říkáme jen *uspořádání*).

`Poznámka`: Pozor, může být relace symetrická i antisymetrická zároveň? Ano!
Vezměte si relaci $R = \{(x, x) \mid x \in M\}$, která obě jmenované vlastnosti splňuje. Proto pokud jste třeba dotázáni, zda relace je symetrická, nestačí se v odpovědi odvolávat na fakt, že je antisymetrická(!), neboť tyto vlastnosti se nevylučují.



Příklad 4.3. *Několik příkladů relací definovaných v přirozeném jazyce.*

Buď M množina všech studentů 1. ročníku FI. Uvažme postupně relace $R \subseteq M \times M$ definované takto

$*$ $(x, y) \in R$ právě když x a y mají stejné rodné číslo;

* $(x, y) \in R$ právě když x má stejnou výšku jako y (dejme tomu na celé mm);
* $(x, y) \in R$ právě když výška x a y se neliší více jak o 2 mm;
* $(x, y) \in R$ právě když x má alespoň takovou výšku jako y;
* $(x, y) \in R$ právě když x má jinou výšku než y (dejme tomu na celé mm);
* $(x, y) \in R$ právě když x je zamilován(a) do y.

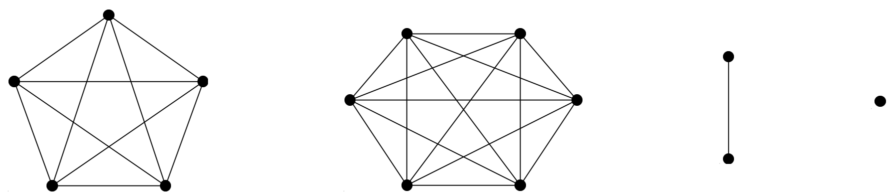
Zamyslete se podrobně, které z definovaných vlastností tyto relace mají. $\square$

Příklad 4.4. *Jaké vlastnosti mají následující relace?*

* Buď $R \subseteq \mathbb{N} \times \mathbb{N}$ definovaná takto $(x, y) \in R$ právě když x dělí y. (*Částečné uspořádání*, ale ne každá dvě čísla jsou porovnatelná.)
* Buď $R \subseteq \mathbb{N} \times \mathbb{N}$ definovaná takto $(x, y) \in R$ právě když x a y mají stejný zbytek po dělení číslem 5. (*Ekvivalence*.)
* Nechť $F = \{f \mid f : \mathbb{N} \to \mathbb{N}\}$ je množina funkcí. Buď $R \subseteq F \times F$ definovaná takto $(f, g) \in R$ právě když $f(x) < g(x)$ pro všechna x. (*Antisymetrická* a *tranzitivní*, ale ne reflexivní – není uspořádání.) $\square$

4.3 Relace ekvivalence

- Relace $R \subseteq M \times M$ je *ekvivalence* právě když R je reflexivní, symetrická a tranzitivní. Tyto tři vlastnosti je tedy třeba ověřit k důkazu toho, že daná relace R je ekvivalence.
- Jak vypadá *graf ekvivalence*?



- Neformálně řečeno: ekvivalence je relace $R \subseteq M \times M$, taková, že $(x, y) \in R$ právě když x a y jsou v nějakém smyslu „stejné“.

  `Komentář:` Buď M množina všech studentů 1. ročníku FI. Uvažme postupně relace $R \subseteq M \times M$ definované takto

  * $(x, y) \in R$ právě když x má stejnou výšku jako y;
  * $(x, y) \in R$ právě když x má stejnou barvu vlasů jako y;
  * $(x, y) \in R$ právě když x, y mají stejnou výšku a stejnou barvu vlasů;
  * $(x, y) \in R$ právě když x, y mají buď stejnou výšku nebo stejnou barvu vlasů.
    (Tato relace obecně není ekvivalence! Proč?)

Příklad 4.5. *Buď $R \subseteq \mathbb{N} \times \mathbb{N}$ binární relace definovaná takto: $(x, y) \in R$ právě když $|x - y|$ je dělitelné třemi.*

V jakém smyslu jsou zde x a y „stejné“? Dávají stejný zbytek po dělení třemi. $\square$

Příklad 4.6. *Buď R binární relace mezi všemi studenty na přednášce IB000 definovaná takto: $(x, y) \in R$ právě když x i y sedí v první lavici.*

Proč se v tomto případě nejedná o relaci ekvivalence? Protože není reflexivní pro studenty sedící v dalších lavicích. (Takže si dávejte dobrý pozor na správné pochopení definic.) □

4.4 Rozklady a jejich vztah k ekvivalencím

Náplní této části je ukázat jiný přirozený pohled na ekvivalence.

Definice 4.7. ***Rozklad.*** Buď M množina.
Rozklad (na) M je množina podmnožin $\mathcal{N} \subseteq 2^M$ splňující následující tři podmínky:

- $\emptyset \notin \mathcal{N}$ (tj. každý prvek $\mathcal{N}$ je neprázdná podmnožina M);
- pokud $A, B \in \mathcal{N}$, pak buď $A = B$ nebo $A \cap B = \emptyset$;
- $\bigcup_{A \in \mathcal{N}} A = M$.

Prvkům $\mathcal{N}$ se také říká *třídy rozkladu*.

`Komentář:`

* Buď $M = \{a, b, c, d\}$. Pak $\mathcal{N} = \{\{a\}, \{b, c\}, \{d\}\}$ je rozklad na M.
* Nechť $A_0 = \{k \in \mathbb{N} \mid k \bmod 3 = 0\}$, $A_1 = \{k \in \mathbb{N} \mid k \bmod 3 = 1\}$,
  $A_2 = \{k \in \mathbb{N} \mid k \bmod 3 = 2\}$.
  Pak $\mathcal{N} = \{A_0, A_1, A_2\}$ je rozklad všech přirozených čísel $\mathbb{N}$ podle zbytkových tříd.
* Každý rozklad $\mathcal{N}$ na M jednoznačně určuje jistou ekvivalenci $R_\mathcal{N}$ na M.

Věta 4.8. *Buď M množina a $\mathcal{N}$ rozklad na M. Nechť $R_\mathcal{N} \subseteq M \times M$ je relace na M definovaná takto*

$$(x, y) \in R_\mathcal{N} \text{ právě když existuje } A \in \mathcal{N} \text{ taková, že } x, y \in A.$$

Pak $R_\mathcal{N}$ je ekvivalence na M.

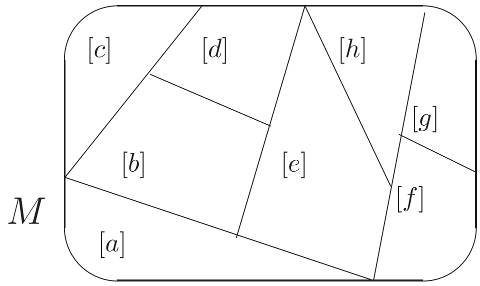
`Důkaz:` Dokážeme, že $R_\mathcal{N}$ je *reflexivní, symetrická a tranzitivní* (*Definice 4.2*).

- Reflexivita: Buď $x \in M$ libovolné. Jelikož $\mathcal{N}$ je rozklad na M, musí existovat $A \in \mathcal{N}$ takové, že $x \in A$ (jinak spor se třetí podmínkou z *Definice 4.7*). Proto $(x, x) \in R_\mathcal{N}$, tedy $R_\mathcal{N}$ je reflexivní.
- Symetrie: Nechť $(x, y) \in R_\mathcal{N}$. Podle definice $R_\mathcal{N}$ pak existuje $A \in \mathcal{N}$ taková, že $x, y \in A$. To ale znamená, že také $(y, x) \in R_\mathcal{N}$ podle definice $R_\mathcal{N}$, tedy $R_\mathcal{N}$ je symetrická.
- Tranzitivita: Nechť $(x, y), (y, z) \in R_\mathcal{N}$. Podle definice $R_\mathcal{N}$ existují $A, B \in \mathcal{N}$ takové, že $x, y \in A$ a $y, z \in B$. Jelikož $A \cap B \neq \emptyset$, podle druhé podmínky z *Definice 4.7* platí $A = B$. Tedy $x, z \in A = B$, proto $(x, z) \in R_\mathcal{N}$ podle definice $R_\mathcal{N}$. □

Věta 4.9. *Buď M množina a R ekvivalence na M. Pro každé $x \in M$ definujeme množinu*

$$[x] = \{y \in M \mid (x, y) \in R\}.$$

Pak $\{[x] \mid x \in M\}$ je rozklad na M, který značíme M/R.



`Důkaz`: Dokážeme, že M/R splňuje podmínky *Definice 4.7.*

- Pro každé $[x] \in M/R$ platí $[x] \neq \emptyset$, neboť $x \in [x]$.
- Nechť $[x], [y] \in M/R$. Ukážeme, že pokud $[x] \cap [y] \neq \emptyset$, pak $[x] = [y]$.

  Jestliže $[x] \cap [y] \neq \emptyset$, existuje $z \in M$ takové, že $z \in [x]$ a $z \in [y]$. Podle definice $[x]$ a $[y]$ to znamená, že $(x, z), (y, z) \in R$. Jelikož R je symetrická a $(y, z) \in R$, platí $(z, y) \in R$. Jelikož $(x, z), (z, y) \in R$ a R je tranzitivní, platí $(x, y) \in R$. Proto také $(y, x) \in R$ (opět ze symetrie R). Nyní dokážeme, že $[y] = [x]$:

  * „$[x] \subseteq [y]$:“ Nechť $v \in [x]$. Pak $(x, v) \in R$ podle definice $[x]$. Dále $(y, x) \in R$ (viz výše), tedy $(y, v) \in R$ neboť R je tranzitivní. To podle definice $[y]$ znamená, že $v \in [y]$.
  * „$[y] \subseteq [x]$:“ Nechť $v \in [y]$. Pak $(y, v) \in R$ podle definice $[y]$. Dále $(x, y) \in R$ (viz výše), tedy $(x, v) \in R$ neboť R je tranzitivní. To podle definice $[x]$ znamená, že $v \in [x]$.

- Platí $\bigcup_{[x] \in M/R} [x] = M$, neboť $x \in [x]$ pro každé $x \in M$. $\square$

Rozšiřující studium

....

5 Uspořádané množiny, Uzávěry

Úvod

V této lekci dále pokračujeme probíráním binárních relací na množinách jako nástroji vyjadřujícími vztahy mezi objekty. Zaměřujeme se nyní především na relace „srovnávající“ objekty podle jejich vlastností. Takto vágně opsané relace mívají jasné společné znaky, které se objevují ve zde uvedené formální definici relace uspořádání.

Následně se také zabýváme způsobem, jak libovolnou relaci „obohatit“ o nějakou vlastnost. Tnto úkol vede na rozšiřování naší relace (tj. přidávání dvojic) do jejího takzvaného uzávěru.

Cíle

Definujeme relace uspořádání a pojmy k nim se vztahující. Dále ukážeme operátory uzávěrů relací.

5.1 Uspořádané množiny

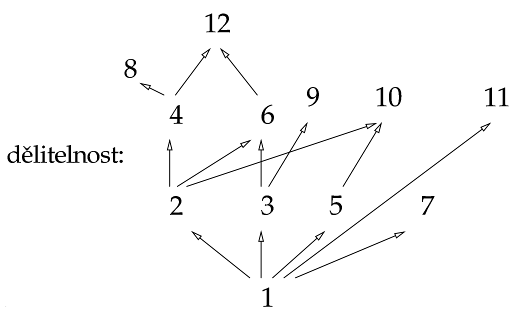
Zopakujme si na úvod...

- Relace $R \subseteq M \times M$ je *(částečné) uspořádání* právě když R je reflexivní, antisymetrická a tranzitivní.Tyto tři vlastnosti je tedy třeba ověřit k důkazu toho, že daná relace R je uspořádání.

- Neformálně řečeno: uspořádání je taková relace $R \subseteq M \times M$, kde $(x, y) \in R$ právě když x je v nějakém smyslu *„menší nebo rovno“* než y. Mohou ovšem existovat taková $x, y \in M$, kde neplatí $(x, y) \in R$ ani $(y, x) \in R$. (Pak říkáme, že x a y jsou *nesrovnatelné*.)

  `Komentář:` Zajisté jste se již neformálně setkali s „neostrým“ uspořádáním čísel $\leq$ a „ostrým“ uspořádáním $<$.Všimněte si dobře, že námi definované uspořádání je vždy „neostré“. Avšak pokud byste chtěli definovat *„ostré“ uspořádání*, mělo by vlastnosti ireflexivní, antisymetrické a tranzitivní. (Příliš se však toto nepoužívá.)

- Jak názorně zobrazit (částečné) uspořádání? Příklad zjednodušeného zakreslení (jsou vynechány šipky vyplývající z reflexivity a tranzitivity, viz Oddíl 5.3):



  Všimněte si, že je zvykem „větší“ prvky kreslit nad ty „menší“.

Definice 5.1. ***Uspořádaná množina*** je dvojice $(M, \sqsubseteq)$,
kde M je množina a $\sqsubseteq$ je (částečné) uspořádání na M.

`Definice`: Uspořádání R na M je *lineární* (nebo také *úplné*), pokud každé dva prvky M jsou v R srovnatelné.

`Komentář:` Buď M množina všech studentů 1. ročníku FI. Uvažme postupně relace uspořádání $R \subseteq M \times M$ definované takto:

* $(x, y) \in R$ právě když x má alespoň takovou výšku jako y;
* $(x, y) \in R$ právě když y má alespoň takovou výšku jako x;
* $(x, y) \in R$ právě když x a y mají stejné rodné číslo. (Dobře si promyslete, proč se jedná o uspořádání. Které dvojice jsou vůbec porovnatelné?)

Další ukázky *uspořádaných množin* následují zde:

* $(\mathbb{N}, \leq)$ je lineárně uspořádaná množina, kde $\leq$ má „obvyklý“ význam.
* $(\mathbb{N}, |)$, kde $|$ je relace dělitelnosti, je uspořádaná množina. Toto uspořádání není lineární.
* Buď M množina. Pak $(2^M, \subseteq)$ je uspořádaná množina (říkáme *inkluzí*).

Příklad 5.2. *Uspořádání „po složkách“.*

Nechť $(A, \leq_A)$ a $(B, \leq_B)$ jsou uspořádané množiny. Definujme binární relaci $\sqsubseteq$ na $A \times B$ předpisem

$$(a, b) \sqsubseteq (a', b') \quad \text{právě když} \quad a \leq_A a' \text{ a } b \leq_B b'.$$

Pak $(A \times B, \sqsubseteq)$ je uspořádaná množina. Toto uspořádání se nazývá „po složkách“. $\square$

Příklad 5.3. *Lexikografické uspořádání.*

Nechť $(A, \leq_A)$ a $(B, \leq_B)$ jsou uspořádané množiny. Definujme binární relaci $\preceq$ na $A \times B$ předpisem

$$(a, b) \preceq (a', b') \quad \text{právě když} \quad \text{buď } a \leq_A a' \text{ a } a \neq a', \text{ nebo } a = a' \text{ a } b \leq_B b'.$$

Pak $(A \times B, \preceq)$ je uspořádaná množina. Navíc pokud $\leq_A$ i $\leq_B$ jsou lineární, je i $\preceq$ lineární. Toto uspořádání se nazývá lexikografické. $\square$

`Fakt`: Jsou-li $(A_1, \leq_1), \cdots, (A_n, \leq_n)$ uspořádané množiny, kde $n \geq 2$, pak množinu $A_1 \times \cdots \times A_n$ lze uspořádat *po složkách* nebo *lexikograficky*.

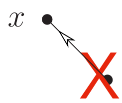
Všimněte si, že lexikograficky se řadí slova ve slovníku...

5.2 Další pojmy uspořádaných množin

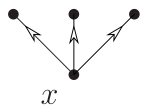
K tématu uspořádaných množin se vztahuje množství drobných pojmů, které potkáte v různých oblastech matematiky i informatiky.

Definice 5.4. Buď $(M, \sqsubseteq)$ uspořádaná množina.

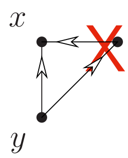
- $x \in M$ je *minimální* právě když pro každé $y \in M$ platí, že jestliže $y \sqsubseteq x$, pak $x \sqsubseteq y$. (Tj. x je minimální právě když neexistuje žádný prvek ostře menší než x.)



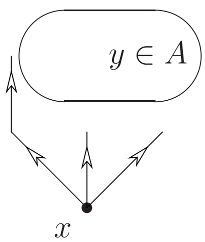
- $x \in M$ je *maximální* právě když pro každé $y \in M$ platí, že jestliže $x \sqsubseteq y$, pak $y \sqsubseteq x$. (Tj. x je maximální právě když neexistuje žádný prvek ostře větší než x.)
- $x \in M$ je *nejmenší* právě když pro každé $y \in M$ platí, že $x \sqsubseteq y$.



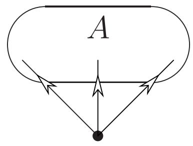
- $x \in M$ je *největší* právě když pro každé $y \in M$ platí, že $y \sqsubseteq x$.
- $x \in M$ *pokrývá* $y \in M$ právě když $x \neq y$, $y \sqsubseteq x$ a neexistuje žádné $z \in M$ takové, že $x \neq z \neq y$ a $y \sqsubseteq z \sqsubseteq x$.



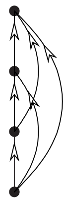
- $x \in M$ je *horní závora* (mez) množiny $A \subseteq M$ právě když $y \sqsubseteq x$ pro každé $y \in A$.
- $x \in M$ je *dolní závora* (mez) množiny $A \subseteq M$ právě když $x \sqsubseteq y$ pro každé $y \in A$.



- $x \in M$ je *supremum* množiny $A \subseteq M$, právě když x je nejmenší horní závora množiny A.
- $x \in M$ je *infimum* množiny $A \subseteq M$ právě když x je největší dolní závora množiny A.



- $A \subseteq M$ je *řetězec* v uspořádání $\sqsubseteq$ právě když $(A, \sqsubseteq)$ je *lineárně* uspořádaná množina.



`Komentář:` Pozor! Některé uvedené definice mají dosti „netriviální chování“ na *nekonečných* množinách. Proto je budeme obvykle uvažovat jen nad konečnými množinami...

Relace předuspořádání

`Definice`: Relace $R \subseteq M \times M$ je *předuspořádání* (také *kvaziuspořádání*, nebo *polouspořádání*) právě když R je *reflexivní a tranzitivní.*

`Komentář:` Rozdíl mezi uspořádáním a předuspořádáním je (neformálně řečeno!) v tom, že u předuspořádání srovnáváme prvky podle kritéria, které není pro daný prvek jedinečné. V předuspořádání takto mohou vznikat „*cykly*“.

Tvrzení 5.5. Je-li $\sqsubseteq$ předuspořádání na M, můžeme definovat relaci $\sim$ na M předpisem

$$x \sim y \quad \text{právě když} \quad x \sqsubseteq y \text{ a } y \sqsubseteq x.$$

Pak $\sim$ je ekvivalence na M, která se nazývá *jádro předuspořádání* $\sqsubseteq$.

Na rozkladu $M/\sim$ pak lze zavést relaci $\preceq$ definovanou takto

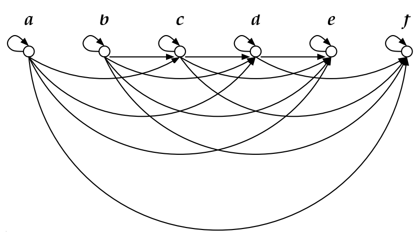
$$[x] \preceq [y] \quad \text{právě když } x \sqsubseteq y.$$

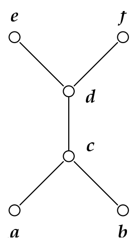
Pak $(M/\sim, \preceq)$ je uspořádaná množina. □

`Komentář:` Pro příklad si vezměme relaci dělitelnosti na $\mathbb{Z}$. Pak třeba $-2 \sim 2$. Jádrem zde jsou dvojice čísel stejné absolutní hodnoty.

5.3 Hasseovské diagramy

Hasseovské diagramy uspořádaných množin jsou přehlednější než grafy relací. Například:



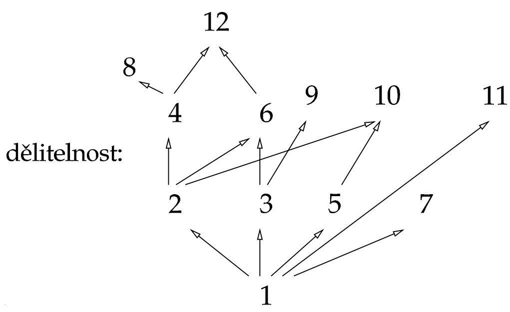


`Definice`: *Hasseovský diagram* konečné uspořádané množiny $(M, \sqsubseteq)$ je (jednoznačné) grafické znázornění vzniklé takto:

– Do první „horizontální vrstvy“ zakreslíme body odpovídající mininálním prvkům $(M, \sqsubseteq)$ (tj. které nepokrývají nic).

– Máme-li již zakreslenu „vrstvu“ i, pak do „vrstvy“ $i + 1$ (která je „nad“ vrstvou i) zakreslíme všechny nezakreslené prvky, které pokrývají pouze prvky „vrstev“ $\leq i$. Pokud prvek x „vrstvy“ $i+1$ pokrývá prvek y „vrstvy“ $\leq i$, spojíme x a y neorientovanou hranou (tj. „čárou“).

Příklad 5.6. *Relaci dělitelnosti na množině* $\{1, 2, \ldots, 12\}$ *zakreslíme:*



□

`Komentář:` Jak vidíme, v Hasseově diagramu „*vynecháváme*“ ty hrany relace $\sqsubseteq$, které vyplývají z *reflexivity či tranzitivity*. To celý obrázek výrazně zpřehlední, a přitom nedochází ke ztrátě informace. Lze vynechat i šipky na hranách, neboť dle definice všechny míří „vzhůru“.

Také pojem „*vrstvy*“ v definici je jen velmi neformální, důležité je, že větší (pokrývající) prvky jsou nad menšími (pokrývanými).

5.4 Uzávěry relací

Buď V (nějaká) vlastnost binárních relací. Řekneme, že V je *uzavíratelná*, pokud splňuje následující podmínky:

– Pro každou množinu M a každou relaci $R \subseteq M \times M$ existuje alespoň jedna relace $S \subseteq M \times M$, která má vlastnost V a pro kterou platí $R \subseteq S$.

– Nechť I je množina a nechť $R_i \subseteq M \times M$ je relace mající vlastnost V pro každé $i \in I$. Pak relace $\bigcap_{i \in I} R_i$ má vlastnost V.

`Fakt`: Libovolná kombinace vlastností *reflexivita, symetrie, tranzitivita* je uzavíratelná vlastnost. Antisymetrie není uzavíratelná vlastnost.

Věta 5.7. *Nechť V je uzavíratelná vlastnost binárních relací. Buď M množina a R libovolná binární relace na M. Pak pro množinu všech relací $S \supseteq R$ na M majících vlastnost V existuje infimum R_V (vzhledem k množinové inkluzi), které samo má vlastnost V.*

Tuto „nejmenší“ relaci R_V s vlastností V nazýváme *V-uzávěr* relace R.

Tvrzení 5.8. Buď R binární relace na M.

∗ *Reflexivní uzávěr* R je přesně relace $R \cup \{(x, x) \mid x \in M\}$.

- ∗ *Symetrický uzávěr* R je přesně relace $\overleftrightarrow{R} = \{(x, y) \mid (x, y) \in R \text{ nebo } (y, x) \in R\}$.

Buď $\mathcal{T}$ funkce, která pro každou binární relaci S vrátí relaci

$$\mathcal{T}(S) = S \cup \{(x, z) \mid \text{ existuje } y \text{ takové, že } (x, y), (y, z) \in S\}$$

a $\mathcal{T}^i = \underbrace{\mathcal{T} \circ \cdots \circ \mathcal{T}}_{i}$ budiž i-krát iterovaná aplikace funkce $\mathcal{T}$.

- ∗ *Tranzitivní uzávěr* R je přesně relace $R^+ = \bigcup_{i=1}^{\infty} \mathcal{T}^i(R)$.
- ∗ Reflexivní a tranzitivní uzávěr R je přesně relace $R^* = \bigcup_{i=1}^{\infty} \mathcal{T}^i(Q)$, kde Q je reflexivní uzávěr R.
- ∗ Reflexivní, symetrický a tranzitivní uzávěr R (tj. nejmenší ekvivalence obsahující R) je přesně relace $(\overleftrightarrow{Q})^+$, kde Q je reflexivní uzávěr R.

`Komentář:` Význam reflexivních a symetrických uzávěrů je z předchozího docela zřejmý. Význam tranzitivního uzávěru R^+ je následovný: Do R^+ přidáme všechny ty dvojice (x, z) takové, že v R se lze „dostat po šipkách“ z x do z. A jak bylo dříve řečeno, antisymetrický uzávěr relace prostě nemá smysl.

Buď $R \subseteq \mathbb{N} \times \mathbb{N}$ definovaná takto: $R = \{(i, i + 1) \mid i \in \mathbb{N}\}$. Pak R^* je běžné $\leq$.

Rozšiřující studium

....

6 Vlastnosti funkcí a Skládání relací

Úvod

Vraťme se nyní k látce Lekce 3. Z jejího pokročilého obsahu jsme doposud velmi detailně probírali relace a jejich jednotlivé vlastnosti. Nyní se podívejme, jak lze relace mezi sebou „skládat“, což je například základní technika práce s relačními databázemi. Je však i jiné místo, kde jste se zajisté se skládáním relací setkali – jedná se o skládání funkcí. Jak například spočítáte na kalkulačce výsledek složitějšího vzorce? Mimo to se ještě zobecněně vrátíme k problematice rekurentních definic (vztahů).

Cíle

V této lekci definujeme základní vlastnosti funkcí a především popíšeme a podrobně rozebereme skládání relací a návazně skládání funkcí jako relací. Na závěr se stručně podíváme na problematiku induktivních definic funkcí, coby na rozšíření rekurentních vztahů z Oddílu 3.5.

6.1 Vlastnosti funkcí

`Definice`: Funkce $f : A \to B$ je

- – *injektivní* (nebo také *prostá*) právě když pro každé $x, y \in A$, $x \neq y$ platí, že $f(x) \neq f(y)$;

– *surjektivní* (nebo také „*na*“) právě když pro každé $y \in B$ existuje $x \in A$ takové, že $f(x) = y$;

– *bijektivní* (vzáj. *jednoznačná*) právě když je injektivní a současně surjektivní.

`Komentář:` Ukázky vlastností funkcí.

* Funkce $plus : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ je surjektivní, ale není prostá.

* Funkce $g : \mathbb{Z} \to \mathbb{N}$ daná předpisem

$$g(x) = \begin{cases} -2x - 1 & \text{jestliže } x < 0, \\ 2x & \text{jinak} \end{cases}$$

  je bijektivní.

* Funkce $\emptyset : \emptyset \to \emptyset$ je bijektivní.

* Funkce $\emptyset : \emptyset \to \{a, b\}$ je injektivní, ale není surjektivní.

6.2 Inverzní relace a skládání relací

`Definice`: Nechť $R \subseteq A \times B$ je binární relace mezi A a B. *Inverzní relace k relaci R* se značí R^{-1} a je definována takto:

$$R^{-1} = \{(b, a) \mid (a, b) \in R\}$$

(R^{-1} je tedy relace mezi B a A.)

`Komentář:` Příklady inverzí pro relace–funkce.

* Inverzí *bijektivní* funkce $f(x) = x + 1$ na $\mathbb{Z}$ je funkce $f^{-1}(x) = x - 1$.

* Inverzí *prosté* funkce $f(x) = e^x$ na $\mathbb{R}$ je *parciální* funkce $f^{-1}(x) = \ln x$.

* Funkce $g(x) = x \bmod 3$ *není prostá* na $\mathbb{N}$, a proto její inverzí je „*jen*“ relace $g^{-1} = \{(a, b) \mid a = b \bmod 3\}$.
  Konkrétně $g^{-1} = \{(0, 0), (0, 3), (0, 6), \dots, (1, 1), (1, 4), \dots, (2, 2), (2, 5), \dots\}$.

Tvrzení 6.1. *Mějme funkci $f : A \to B$. Pak její inverzní relace f^{-1} je*

a) parciální funkce právě když f je prostá,

b) funkce právě když f je bijektivní.

Důkaz vyplývá přímo z definic funkce a inverze relace. □

Definice 6.2. ***Složení (kompozice)*** relací R a S.
Nechť $R \subseteq A \times B$ a $S \subseteq B \times C$ jsou binární relace. *Složení* relací R a S (v tomto pořadí!) je relace $S \circ R \subseteq A \times C$ definovaná takto:

$$S \circ R = \{(a, c) \mid \text{existuje } b \in B \text{ takové, že } (a, b) \in R, (b, c) \in S\}$$

Složení relací čteme „*R složeno s S*“ nebo (pozor na pořadí!) „*S po R*“.

`Komentář:` Příklady skládání relací.

* Je-li

– $A = \{a, b\}, \quad B = \{1, 2\}, \quad C = \{X\},$
– $R = \{(a, 1), (b, 1), (b, 2)\}, \quad S = \{(1, X)\},$

pak složením vznikne relace

– $S \circ R = \{(a, X), (b, X)\}.$

∗ Složením funkcí $h(x) = x^2$ a $f(x) = x + 1$ na $\mathbb{R}$ vznikne funkce

$$f \circ h\,(x) = f(h(x)) \;=\; x^2 + 1.$$

∗ Složením těchže funkcí „*naopak*“ ale vznikne funkce $h \circ f\,(x) = h(f(x)) \;=\; (x + 1)^2$.

`Poznámka`: Nepříjemné je, že v některých oblastech matematiky (například v algebře při skládání zobrazení) se setkáme s právě opačným zápisem skládání, kdy se místo $S \circ R$ píše $R \cdot S$ nebo jen RS. Proto je si vždy dobré slovně ujasnit, které pořadí skládaných relací máme na mysli. My zde zásadně budeme používat pořadí $S \circ R$.

6.3 Skládání relací „v praxi“

Podívejme se nyní, jak se skládání relací přirozeně objevuje v práci s relačními databázemi. (Dá se zjednodušeně říci, že právě v operátoru skládání tabulkových relací je hlavní smysl relačních databází…)

Příklad 6.3. *Skládání v relační databázi studentů, jejich předmětů a fakult.*

Mějme dvě binární relace – jednu R přiřazující studentům MU kódy jejich zapsaných předmětů, druhou S přiřazující kódy předmětů jejich mateřským fakultám. Malý výsek z těchto relací může v tabulkové reprezentaci vypadat třeba následovně.

R :

student (učo)	předmět (kód)
121334	MA010
133935	M4135
133935	IA102
155878	M1050
155878	IB000

S :

předmět (kód)	fakulta MU
MA010	FI
IB000	FI
IA102	FI
M1050	PřF
M4135	PřF

Jak z těchto „tabulkových“ relací zjistíme, kteří studenti mají *zapsané předměty na kterých fakultách* (třeba na FI)?

Jedná se jednoduše o složení relací $S \circ R$. V našem příkladě tabulkové reprezentace vyjde výsek:

$S \circ R$:

student (učo)	fakulta MU
121334	FI
133935	FI
133935	PřF
155878	FI
155878	PřF

□

Zobecněné skládání relací

V praktických použitích relačních tabulek povětšinou nevystačíme jen s binárními relacemi, takže je přirozené se ptát, jestli lze podobně skládat i více-ární relace. Odpověď je snadná – lze to a ani nepotřebujeme novou definici, vystačíme s tou, kterou už máme výše uvedenou.

`Fakt` (***skládání relací vyšší arity***):
Mějme relace $T \subseteq K_1 \times K_2 \times \ldots \times K_k$ a $U \subseteq L_1 \times L_2 \times \ldots \times L_\ell$, přičemž pro nějaké $m < \min(k, \ell)$ platí $L_1 = K_{k-m+1}, L_2 = K_{k-m+2}, \ldots, L_m = K_k$. Pak relaci T lze *složit* s relací U na zvolených m složkách $L_1, \ldots, L_m$ („překrytí“) s *použitím Definice 6.2* takto:

* Položme $A = K_1 \times \ldots \times K_{k-m}$, $B = L_1 \times \ldots \times L_m$ a $C = L_{m+1} \times \ldots \times L_\ell$.
* Příslušné relace pak jsou $R = \{(\vec{a}, \vec{b}) \in A \times B \mid (a_1, \ldots a_{k-m}, b_1, \ldots b_m) \in T\}$ a $S = \{(\vec{b}, \vec{c}) \in B \times C \mid (b_1, \ldots b_m, c_{m+1}, \ldots c_\ell) \in U\}$.
* Nakonec *přirozeně položme* $U \circ_m T \simeq A \circ B$, takže vyjde

$$U \circ_m T = \{(\vec{a}, \vec{c}) \mid \text{ ex. } \vec{b} \in B, \text{ že } (a_1, \ldots a_{k-m}, b_1, \ldots b_m) \in T \text{ a } (b_1, \ldots b_m, c_{m+1}, \ldots c_\ell) \in U\}.$$

Schematicky pro snažší orientaci ve složkách našich relací:

$$\begin{array}{rcccc}
T \subseteq & K_1 \times \ldots \times K_{k-m} \times & K_{k-m+1} \times \ldots \times K_k & & \\
U \subseteq & & L_1 \times \ldots \times L_m & \times L_{m+1} \times \ldots \times L_\ell \\
U \circ_m T \subseteq & \underbrace{K_1 \times \ldots \times K_{k-m}}_{A} \times & \underbrace{\qquad\qquad\qquad}_{B} & \times \underbrace{L_{m+1} \times \ldots \times L_\ell}_{C}
\end{array}$$

Opět je nejjednodušší si koncept skládání vícečetných relací ilustrovat příkladem.

Příklad 6.4. *Skládání v relační databázi pasažérů a letů u leteckých společností.*

Podívejme se na příklad hypotetické rezervace letů pro cestující, relace T. Jak známo (tzv. codeshare), letecké společnosti si mezi sebou „dělí“ místa v letadlech, takže různé lety (podle kódů) jsou ve skutečnosti realizovány stejným letadlem jedné ze společností. To zase ukazuje relace U.

T :

pasažér	datum	let
Petr	5.11.	OK535
Pavel	6.11.	OK535
Jan	5.11.	AF2378
Josef	5.11.	DL5457
Alena	6.11.	AF2378

U :

datum	let	letadlo
5.11.	OK535	ČSA
5.11.	AF2378	ČSA
5.11.	DL5457	ČSA
6.11.	OK535	AirFrance
6.11.	AF2378	AirFrance

Ptáme-li se nyní, setkají se Petr a Josef na palubě stejného letadla? Případně, čí letadlo to bude? Odpovědi nám dá zase složení relací $U \circ_2 T$, jak je posáno výše.

$U \circ_2 T$:

pasažér	letadlo
Petr	ČSA
Josef	ČSA
...	...

Zkuste se zamyslet, lze tyto dvě relace skládat ještě jinak? Co by pak bylo významem? □

6.4 Skládání funkcí, permutace

Soustřeďme se nyní na další oblast, kde běžně a přirozeně používáme skládání relací, aniž si to uvědomujeme.

`Fakt`: Mějme zobrazení (funkce) $f : A \to B$ a $g : B \to C$. Pak jejich *složením* coby relací v tomto pořadí vznikne zobrazení $(g \circ f) : A \to C$ definované

$$(g \circ f)(x) = g(f(x)) \,.$$

`Komentář:`

* Jak například na běžné kalkulačce vypočteme hodnotu funkce $\sin^2 x$? Složíme (v tomto pořadí) „elementární“ funkce $f(x) = \sin x$ a $g(x) = x^2$.
* Jak bychom na „elementární“ funkce rozložili aritmetický výraz $2\log(x^2+1)$? Ve správném pořadí složíme funkce $f_1(x) = x^2$, $f_2(x) = x+1$, $f_3(x) = \log x$ a $f_4(x) = 2x$.
* A jak bychom obdobně vyjádřili složením funkcí aritmetický výraz $\sin x + \cos x$? Opět je odpověď přímočará, vezmeme „elementární“ funkce $g_1(x) = \sin x$ a $g_2(x) = \cos x$, a pak je „složíme“ další funkcí $h(x,y) = x+y$. Vidíme však, že takto pojaté „skládání“ už nezapadá hladce do našeho formalismu skládání relací.

  Pro nedostatek prostoru si skládání funkcí s více parametry nedefinujeme, ale sami vidíte, že obdobné skládání se v programátorské praxi vyskytuje doslova „na každém rohu“ a ani se nad tím nepozastavujeme.

Skládání permutací

Po zbytek tohoto oddílu se zaměříme na permutace coby speciální případ (bijektivních) zobrazení.

`Definice`: Nechť *permutace* π množiny $[1,n]$ je určena seřazením jejích prvků $(p_1, \dots, p_n)$. Pak π je zároveň *bijektivním zobrazením* $[1,n] \to [1,n]$ definovaným předpisem $\pi(i) = p_i$.
Tudíž lze permutace *skládat jako relace* podle Definice 6.2.

`Poznámka`: Všechny permutace množiny $[1,n]$ spolu s operací skládání tvoří grupu, zvanou symetrická grupa S_n. Permutační grupy (podgrupy symetrické grupy) jsou velice důležité v algebře, neboť každá grupa je vlastně isomorfní některé permutační grupě.

`Komentář:` Příkladem permutace vyskytujícím se v programátorské praxi je třeba zobrazení $i \mapsto (i+1) \bmod n$.
Často se programátor setkává (aniž si to mnohdy uvědomuje) s permutacemi při indexaci prvků polí.

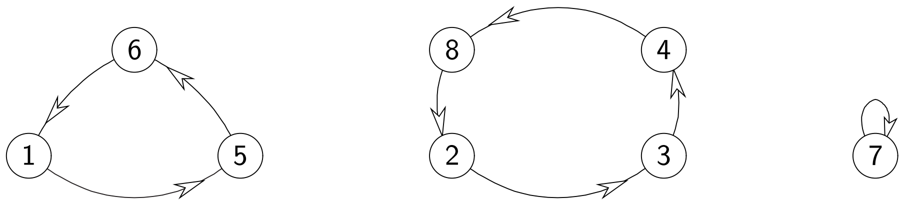
V kontextu pohledu na funkce a jejich skládání coby relací si zavedeme jiný, názornější, způsob zápisu permutací – pomocí jejich cyklů.

`Definice`: Nechť π je permutace na množině A. *Cyklem v π* rozumíme posloupnost $\langle a_1, a_2, \dots, a_k \rangle$ různých prvků A takovou, že $\pi(a_i) = a_{i+1}$ pro $i = 1, 2, \dots, k-1$ a $\pi(a_k) = a_1$.

Jak název napovídá, v zápise cyklu $\langle a_1, a_2, \dots, a_k \rangle$ není důležité, kterým prvkem začneme, ale jen dodržení cyklického pořadí. Cyklus v permutaci může mít i jen jeden prvek (zobrazený na sebe).

`Komentář:` Nakreslete si (vámi zvolenou) permutaci π obrázkem, ve kterém vedete šipku vždy od prvku i k prvku $\pi(i)$. Pak uvidíte, že cykly dle naší definice jsou právě cykly tvořené šipkami ve vašem obrázku. S tímto grafickým zobrazením pro vás nebude problém pochopit následující tvrzení.

Například permutaci $(5, 3, 4, 8, 6, 1, 7, 2)$ si lze obrázkem nakreslit takto:



Věta 6.5. *Každou permutaci π na konečné množině A lze zapsat jako složení cyklů na disjunktních podmnožinách (rozkladu) A.*

`Důkaz`: Vezmeme libovolný prvek $a_1 \in A$ a iterujeme zobrazení $a_2 = \pi(a_1)$, $a_3 = \pi(a_2)$, atd., až se dostaneme „zpět“ k $a_{k+1} = \pi(a_k) = a_1$. Proč tento proces skončí? Protože A je konečná a tudíž ke zopakování některého prvku a_{k+1} musí dojít. Nadto je π prostá, a proto nemůže nastat $\pi(a_k) = a_j$ pro $j > 1$. Takto získáme první cyklus $\langle a_1, \dots, a_k \rangle$.

Induktivně pokračujeme s hledáním dalších cyklů ve zbylé množině $A \setminus \{a_1, \dots, a_k\}$, dokud nezůstane prázdná. □

`Značení` permutací cykly: Nechť se permutace π podle Věty 6.5 skládá z cyklů $\langle a_1, \dots, a_k \rangle$, $\langle b_1, \dots, b_l \rangle$ až třeba $\langle z_1, \dots, z_m \rangle$. Pak zapíšeme

$$\pi = (\langle a_1, \dots, a_k \rangle \langle b_1, \dots, b_l \rangle \dots \langle z_1, \dots, z_m \rangle) \, .$$

`Komentář:` Primitivní pseudonáhodné generátory v počítačích iterují z náhodného počátku permutaci danou vztahem $i \mapsto (i + p) \bmod q$. Je pochopitelné, že tato permutace nesmí obsahovat krátké cykly, lépe řečeno, měla by se skládat z jediného (dlouhého) cyklu. (Pro úplnost, jedná se o permutaci množiny $\{0, 1, \dots, q - 1\}$).

Příklad 6.6. *Ukázka skládání permutací daných svými cykly.*

Vezměme 7-prvkovou permutaci $(5, 3, 4, 2, 6, 1, 7)$. Ta se rozkládá na tři cykly $\langle 1, 5, 6 \rangle$, $\langle 2, 3, 4 \rangle$ a $\langle 7 \rangle$. Jiná permutace $(3, 4, 5, 6, 7, 1, 2)$ se skládá z jediného cyklu $\langle 1, 3, 5, 7, 2, 4, 6 \rangle$.

Nyní určíme složení těchto dvou permutací (zápisem cykly):

$$(\langle 1, 5, 6 \rangle \langle 2, 3, 4 \rangle \langle 7 \rangle) \circ (\langle 1, 3, 5, 7, 2, 4, 6 \rangle) = (\langle 1, 4 \rangle \langle 2 \rangle \langle 3, 6, 5, 7 \rangle)$$

(Nezapomínejme, že první se ve složení aplikuje pravá permutace!)
Postup skládání jsme použili následovný: 1 se zobrazí v permutaci vpravo na 3 a pak vlevo na 4. Následně 4 se zobrazí na 6 a pak na 1. Tím „uzavřeme“ první cyklus $\langle 1, 4 \rangle$. Dále se 2 zobrazí na 4 a pak hned zpět na 2, tj. má samostatný cyklus. Zbylý cyklus $\langle 3, 6, 5, 7 \rangle$ určíme analogicky. □

6.5 Induktivní definice množin a funkcí

Vzpomeňme si na definici posloupnosti *rekurentním vztahem* z Oddílu 3.5. Přímým zobecněním rekurentních definic je následující koncept.

Definice 6.7. ***Induktivní definice*** množiny.
Jedná se obecně o popis (nějaké) množiny M v následujícím tvaru:

- Je dáno několik pevných (*bázických*) prvků $a_1, a_2, \ldots, a_k \in M$.
- Je dán soubor *induktivních pravidel* typu

  Jsou-li (libovolné prvky) $x_1, \ldots, x_\ell \in M$, pak také $y \in M$.

  V tomto případě je y typicky funkcí $y = f_i(x_1, \ldots, x_\ell)$.

Pak naše *induktivně definovaná množina* M je určena jako nejmenší (inkluzí) množina vyhovující těmto pravidlům.

`Komentář:` Vidíte podobnost této definice s uzávěrem relace? (Věta 5.7.)
Pro nejbližší příklad induktivní definice se obrátíme na množinu všech přirozených čísel.
- $0 \in \mathbb{N}$
- Je-li $i \in \mathbb{N}$, pak také $i + 1 \in \mathbb{N}$.

Pro každé $y \in \mathbb{N}$ můžeme definovat jinou množinu $M_y \subseteq \mathbb{N}$ induktivně takto:
- $y \in M_y$.
- Jestliže $x \in M_y$ a $x + 1$ je liché, pak $x + 2 \in M_y$.

Pak například $M_3 = \{3\}$, nebo $M_4 = \{4 + 2i \mid i \in \mathbb{N}\}$.

`Definice`: Řekneme, že daná induktivní definice množiny M je *jednoznačná*, právě když každý prvek M lze odvodit z bázových prvků pomocí induktivních pravidel právě jedním způsobem.

`Komentář:` Definujme množinu $M \subseteq \mathbb{N}$ induktivně takto:
- $2, 3 \in M$.
- Jestliže $x, y \in M$, pak také $x^2 + y^2$ a $x.y$ jsou prvky M.

Proč tato induktivní definice není jednoznačná? Například číslo $8 \in M$ lze odvodit způsobem $8 = 2 \cdot (2 \cdot 2)$, ale zároveň zcela jinak $8 = 2^2 + 2^2$.
V čem tedy spočívá důležitost jednoznačných induktivních definic množin?

Definice 6.8. ***Induktivní definice funkce*** z induktivní množiny.
Nechť množina M je dána jednoznačnou induktivní definicí. Pak říkáme, že funkce $\mathcal{F} : M \to X$ je definována *induktivně* (vzhledem k induktivní definici M), pokud je řečeno:

- Pro každý z bázických prvků $a_1, a_2, \ldots, a_k \in M$ je určeno $\mathcal{F}(a_i) = c_i$, kde c_i je konstanta.
- Pro každé induktivní pravidlo typu

  "Jsou-li (libovolné prvky) $x_1, \ldots, x_\ell \in M$, pak také $f(x_1, \ldots, x_\ell) \in M$"

  je definováno

  $\mathcal{F}(\, f(x_1, \ldots, x_\ell)\,)$ na základě hodnot $\mathcal{F}(x_1), \ldots, \mathcal{F}(x_\ell)$.

`Komentář:` Pro příklad se podívejme třeba do manuálových stránek unixového příkazu `test EXPRESSION`:

```
EXPRESSION is true or false and sets exit status.  It is one of:
  ! EXPRESSION                    EXPRESSION is false
  EXPRESSION1 -a EXPRESSION2      both EXPRESSION1 and EXPRESSION2 are true
  EXPRESSION1 -o EXPRESSION2      either EXPRESSION1 or EXPRESSION2 is true
  [-n] STRING                     the length of STRING is nonzero
  STRING1 = STRING2               the strings are equal
     ......
```

Induktivní definice se „strukturální“ indukcí

Příklad 6.9. *Jednoduché aritmetické výrazy*

Nechť (abeceda) $\Sigma = \{0,1,2,3,4,5,6,7,8,9,\odot,\oplus,(,)\}$. Definujme množinu *jednoduchých výrazů* $\mathit{SExp} \subseteq \Sigma^*$ induktivně takto:

– Dekadický zápis každého přirozeného čísla $\mathbf{n}$ je prvek SExp.

– Jestliže $x, y \in \mathit{SExp}$, pak také $(x) \odot (y)$ a $(x) \oplus (y)$ jsou prvky SExp.

(Jak vidíme, díky „závorkování“ je tato induktivní definice jednoznačná.)

Pro „vyhodnocení“ výrazu pak definujme funkci $\mathit{Val} : \mathit{SExp} \to \mathbb{N}$ induktivně takto:

– Bázické prvky: $\mathit{Val}(\mathbf{n}) = n$, kde $\mathbf{n}$ je dekadický zápis přirozeného čísla n.

– První induktivní pravidlo: $\mathit{Val}((x) \oplus (y)) = \mathit{Val}(x) + \mathit{Val}(y)$.

– Druhé induktivní pravidlo: $\mathit{Val}((x) \odot (y)) = \mathit{Val}(x) \cdot \mathit{Val}(y)$.

(Tímto způsobem jsme našim výrazům vlastně přiřadili jejich „význam“, sémantiku.) □

Příklad 6.10. *Důkaz správnosti přiřazeného „významu“* $\mathit{Val} : \mathit{SExp} \to \mathbb{N}$.

> **Věta**. *Pro každý výraz $s \in \mathit{SExp}$ je hodnota $\mathit{Val}(s)$ číselně rovna výsledku vyhodnocení výrazu s podle běžných zvyklostí aritmetiky.*

Jelikož pojednáváme o induktivně definované funkci Val, je přirozené pro důkaz jejích vlastností aplikovat *matematickou indukci*. Na rozdíl od dříve probíraných příkladů zde nevidíme žádný celočíselný „parametr n“, a proto si jej budeme muset nejprve definovat. Naši indukci tedy povedeme podle „délky ℓ odvození výrazu s“ definované jako počet aplikací induktivních pravidel potřebných k odvození $s \in \mathit{SExp}$. Takto aplikované matematické indukci se často říká *strukturální indukce*.

`Důkaz`: V *bázi indukce* ověříme vyhodnocení bázických prvků, což jsou zde dekadizké zápisy přirozených čísel. Platí $\mathit{Val}(\mathbf{n}) = n$, což skutečně odpovídá zvyklostem aritmetiky.

V *indukčním kroku* se podíváme na vyhodnocení $\mathit{Val}((x) \oplus (y)) = \mathit{Val}(x) + \mathit{Val}(y)$. Podle běžných zvyklostí aritmetiky by hodnota $\mathit{Val}((x) \oplus (y))$ měla být rovna *součtu* vyhodnocení *výrazu* x, což je podle indukčního předpokladu rovno $\mathit{Val}(x)$ (x má zřejmě kratší délku odvození), a vyhodnocení *výrazu* y, což je podle indukčního předpokladu rovno $\mathit{Val}(y)$. Takže skutečně $\mathit{Val}((x) \oplus (y)) = \mathit{Val}(x) + \mathit{Val}(y)$. Druhé pravidlo $\mathit{Val}((x) \odot (y))$ se dořeší analogicky. □

Rozšiřující studium

....

7 Jemný úvod do Logiky

Úvod

Základem přesného matematického vyjadřování je správné používání (matematické) logiky a logických úsudků. Logika jako filozofická disciplína se intenzivně vyvíjí už od dob antiky, avšak ke skutečnému rozmachu logiky coby součásti matematiky došlo až začátkem 20. století. (S přispěním třeba Russelova paradoxu.)

Dnes se samozřejmě základní logický kalkulus používá nejen v matematice, ale „stojí" na něm veškeré logické obvody a počítače. Proto se také studenti informatiky s logikou setkávají záhy při svém studiu a mnohokrát se k tématu také vracejí.

Cíle

Následující lekce přináší (skutečně velmi jemný a trochu i povrchní) úvod do moderní matematické logiky, která je solidním základem matematiky a nakonec i celé informatiky. Zavedeme si základy výrokové logiky a výrokového počtu a velmi stručně si uvedeme problematiku predikátové logiky a kvantifikace.

7.1 Výroky v „přirozené" podobě

`Definice`: V přirozené mluvě za *výrok* považujeme (každé) tvrzení, o kterém má smysl prohlásit, že je buď pravdivé nebo nepravdivé.

`Komentář:` Několik příkladů, které z nich jsou výroky?

* Dnes už v Brně pršelo.
* Předmět IB000 se vyučuje v prvním ročníku.
* Platí $2 + 3 = 6$.
* To je bez problémů. *(Co?)*
* Platí $x > 3$.
* Pro každé celé číslo x platí, že $x > 3$.

Všimněte si, že pravdivost výroku by mělo být možné rozhodnout bez skrytých souvislostí (kontextu), a proto čtvrtý a pátý příklad za výroky nepovažujeme.

`Fakt`: Z jednoduchých výroků můžeme vytvářet výroky složitější pomocí tzv. *logických spojek*.

`Komentář:` Několik dalších příkladů.

* Kateřina přijede ve 12:00 a půjdeme spolu do kina.
* Množina $\{a, b\}$ má více než jeden prvek a není nekonečná.
* Jestliže má Karel přes 90 kilo váhy, nepojedu s ním výtahem.
* Jestliže má kráva 10 nohou, mají všechny domy modrou střechu.

Schopnost porozumět takovýmto větám je součást lidského způsobu uvažování a z tohoto hlediska nemá přímou souvislost s matematikou (je to „přirozená logika").

Formální (matematická) logika pak definuje jazyk matematiky a odstraňuje nejednoznačnosti přirozeného jazyka.

7.2 (Formální) výroková logika

Definice 7.1. ***Syntaxe výrokové logiky.***
Buď $\mathcal{AT} = \{A, B, C, \ldots\}$ spočetně nekonečná množina *výrokových proměnných* (tzv. atomů). Množina *výrokových formulí* Φ je *definována induktivně* následujícími pravidly:

(1) $\mathcal{AT} \subseteq \Phi$.

(2) Jestliže $\varphi, \psi \in \Phi$, pak také $\neg(\varphi) \in \Phi$ a $(\varphi) \Rightarrow (\psi) \in \Phi$.

(3) Každý prvek Φ vznikne konečně mnoha aplikacemi pravidel (1) a (2).

`Značení`: Symbol $\neg$ je zván *negací* a $\Rightarrow$ je nazýván *implikací*.

`Komentář`: Příklady několika správně utvořených formulí:

$$A, \quad (A) \Rightarrow (B), \quad ((A) \Rightarrow (\neg(B))) \Rightarrow ((\neg(B)) \Rightarrow (C))$$

A také příklady několika ne zcela správně utvořených formulí:

$$A \Rightarrow B, \quad A \Rightarrow B \Rightarrow C, \quad \neg A \Rightarrow B$$

Konvence 7.2. Pro zvýšení čitelnosti budeme závorky vynechávat, pokud to nepovede k nejednoznačnostem za předpokladu, že negace $\neg$ má „vyšší prioritu“ než $\Rightarrow$. (Touto úmluvou se nemění množina Φ; mění se jen *způsob reprezentace* jejích prvků.

Dále si zavedeme, že

* $\varphi \vee \psi$ *(disjunkce) je jiný zápis formule* $\neg\varphi \Rightarrow \psi$,
* $\varphi \wedge \psi$ *(konjunkce) je jiný zápis formule* $\neg(\neg\varphi \vee \neg\psi)$,
* $\varphi \Leftrightarrow \psi$ *(ekvivalence) je jiný zápis formule* $(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$.

`Komentář`: Například formule $(\neg((A) \Rightarrow (B))) \Rightarrow ((\neg(B)) \Rightarrow (C))$ se dá s naší konvencí zapsat jako

$$(A \Rightarrow B) \vee B \vee C\,.$$

Definice 7.3. ***Sémantika výrokové logiky.***
Valuace (ohodnocení) je funkce $\nu : \mathcal{AT} \to \{true, false\}$.Pro každou valuaci ν definujeme funkci $\mathcal{S}_\nu : \Phi \to \{true, false\}$ (vyhodnocení) induktivně takto:

* $\mathcal{S}_\nu(A) = \nu(A)$ pro každé $A \in \mathcal{AT}$.
* $\mathcal{S}_\nu(\neg\varphi) = \begin{cases} true & \text{jestliže } \mathcal{S}_\nu(\varphi) = false; \\ false & \text{jinak.} \end{cases}$
* $\mathcal{S}_\nu(\varphi \Rightarrow \psi) = \begin{cases} false & \text{jestliže } \mathcal{S}_\nu(\varphi) = true \text{ a } \mathcal{S}_\nu(\psi) = false; \\ true & \text{jinak.} \end{cases}$

Tvrzení 7.4. *Důsledkem této definice je následovné:*

* $\mathcal{S}_\nu(\varphi \vee \psi) = true$ *právě když* $\mathcal{S}_\nu(\varphi) = true$ *nebo* $\mathcal{S}_\nu(\psi) = true$.
* $\mathcal{S}_\nu(\varphi \wedge \psi) = true$ *právě když* $\mathcal{S}_\nu(\varphi) = true$ *a současně* $\mathcal{S}_\nu(\psi) = true$.
* $\mathcal{S}_\nu(\varphi \Leftrightarrow \psi) = true$ *právě když platí jedna z následujících podmínek*
  - $\mathcal{S}_\nu(\varphi) = true$ *a současně* $\mathcal{S}_\nu(\psi) = true$,
  - $\mathcal{S}_\nu(\varphi) = false$ *a současně* $\mathcal{S}_\nu(\psi) = false$.

`Poznámka`: Tento předpis podává nejen *definici* funkce $\mathcal{S}_\nu$, ale také návod na to, jak ji pro daný argument *vypočítat*.

Pravdivostní hodnoty

V praxi vyhodnocení logické (výrokové) formule zapisujeme často tzv. *pravdivostní tabulkou*. Tabulka typicky má sloupce pro jednotlivé proměnné, případné „meziformule" a výslednou formuli. Řádků je pak 2^p, kde p je počet použitých proměnných. Místo *true*, *false* píšeme $1, 0$.

Příklad 7.5. *Jaká je pravdivostní tabulka pro formuli* $(A \Rightarrow B) \vee B \vee C$*?*

A	B	C	$A \Rightarrow B$	$(A \Rightarrow B) \vee B \vee C$
0	0	0	1	1
0	1	0	1	1
1	0	0	0	0
1	1	0	1	1
0	0	1	1	1
0	1	1	1	1
1	0	1	0	1
1	1	1	1	1

□

`Definice`: Formule $\varphi \in \Phi$ je *splnitelná*, pokud pro některou valuaci ν platí, že $\mathcal{S}_\nu(\varphi) = true$.

Formule $\varphi \in \Phi$ je *vždy pravdivá* (také výroková *tautologie*), psáno $\models \varphi$, pokud pro každou valuaci ν platí, že $\mathcal{S}_\nu(\varphi) = true$.

Řekneme, že dvě formule $\varphi, \psi \in \Phi$ jsou *ekvivalentní*, právě když $\models \varphi \Leftrightarrow \psi$.

Tvrzení 7.6. *Několik užitečných tautologií:*

* $\models A \vee \neg A$
* $\models \neg\neg A \Leftrightarrow A$
* $\models (A \wedge (A \Rightarrow B)) \Rightarrow B$
* $\models (\neg B \Rightarrow \neg A) \Rightarrow (A \Rightarrow B)$
* $\models (\neg A \Rightarrow (B \wedge \neg B)) \Rightarrow A$

7.3 Jak správně „znegovat formuli"?

Přesný význam formulí se zanořenými negacemi je někdy obtížné zjistit (podobně jako v běžné řeči).

„Není pravda, že nemohu neříct, že není pravda, že tě nemám nerad."

Výrokové formule se proto obvykle prezentují v *normálním tvaru*, kde se negace vyskytují pouze u výrokových proměnných.

`Komentář`: Například, pokud přijmeme pravidlo „dvojí negace" ($\neg\neg A \Leftrightarrow A$), tak výše napsanou větu si převedeme na lépe srozumitelný tvar:

„Nemusím říct, že tě mám nerad."

Tvrzení 7.7. *Každou výrokou formuli lze převést do normálního tvaru, pokud povolíme užívání odvozených spojek $\wedge$ a $\vee$.*

„Znegováním formule φ“ se obvykle myslí převod $\neg\varphi$ do normálního tvaru.

`Komentář:` Pro ilustraci $\neg(A \Rightarrow B)$ je ekvivalentní $A \wedge \neg B$, $\neg(C \wedge (\neg A \Rightarrow B))$ je ekvivalentní $\neg C \vee (\neg A \wedge \neg B)$.

Formální postup negace

Metoda 7.8. ***Převod formule Φ do normálního tvaru $\mathcal{F}(\Phi)$.***
Definujeme funkce $\mathcal{F}$ a $\mathcal{G}$ pro náš převod induktivními předpisy

$$
\begin{array}{lcllcl}
\mathcal{F}(A) & = & A & \mathcal{G}(A) & = & \neg A \\
\mathcal{F}(\neg\varphi) & = & \mathcal{G}(\varphi) & \mathcal{G}(\neg\varphi) & = & \mathcal{F}(\varphi) \\
\mathcal{F}(\varphi \Rightarrow \psi) & = & \mathcal{F}(\varphi) \Rightarrow \mathcal{F}(\psi) & \mathcal{G}(\varphi \Rightarrow \psi) & = & \mathcal{F}(\varphi) \wedge \mathcal{G}(\psi) \\
\mathcal{F}(\varphi \wedge \psi) & = & \mathcal{F}(\varphi) \wedge \mathcal{F}(\psi) & \mathcal{G}(\varphi \wedge \psi) & = & \mathcal{G}(\varphi) \vee \mathcal{G}(\psi) \\
\mathcal{F}(\varphi \vee \psi) & = & \mathcal{F}(\varphi) \vee \mathcal{F}(\psi) & \mathcal{G}(\varphi \vee \psi) & = & \mathcal{G}(\varphi) \wedge \mathcal{G}(\psi) \\
\mathcal{F}(\varphi \Leftrightarrow \psi) & = & \mathcal{F}(\varphi) \Leftrightarrow \mathcal{F}(\psi) & \mathcal{G}(\varphi \Leftrightarrow \psi) & = & (\mathcal{F}(\varphi) \wedge \mathcal{G}(\psi)) \vee (\mathcal{G}(\varphi) \wedge \mathcal{F}(\psi))
\end{array}
$$

`Komentář:` Uvažme formuli $\neg(A \Rightarrow \neg(B \vee \neg(C \Rightarrow \neg A)))$. Užitím postupu 7.8 získáme:

$$
\begin{array}{lcll}
\mathcal{F}(\neg(A \Rightarrow \neg(B \vee \neg(C \Rightarrow \neg A)))) & = & \mathcal{G}(A \Rightarrow \neg(B \vee \neg(C \Rightarrow \neg A))) & = \\
\mathcal{F}(A) \wedge \mathcal{G}(\neg(B \vee \neg(C \Rightarrow \neg A))) & = & A \wedge \mathcal{F}(B \vee \neg(C \Rightarrow \neg A)) & = \\
A \wedge (\mathcal{F}(B) \vee \mathcal{F}(\neg(C \Rightarrow \neg A))) & = & A \wedge (B \vee \mathcal{G}(C \Rightarrow \neg A)) & = \\
A \wedge (B \vee (\mathcal{F}(C) \wedge \mathcal{G}(\neg A))) & = & A \wedge (B \vee (C \wedge \mathcal{F}(A))) & = \\
A \wedge (B \vee (C \wedge A)) & & &
\end{array}
$$

Formuli $A \wedge (B \vee (C \wedge A))$ lze dále zjednodušit na (ekvivalentní) formuli $A \wedge (B \vee C)$. To ale je již z našeho pohledu matematicky neformální (heuristický) postup.

Uvedené formální předpisy takto vyjadřují „intuitivní postup negace“ v matematicky přesném tvaru.

Věta 7.9. *Pro libovolnou výrokovou formuli φ platí, že*

a) $\mathcal{F}(\varphi)$ je jí ekvivalentní formule v normálním tvaru

b) a $\mathcal{G}(\varphi)$ je formule v normálním tvaru ekvivalentní negaci $\neg\varphi$.

`Důkaz` povedeme tzv. „*indukcí ke struktuře formule*“, tedy indukci povedeme podle „délky“ ℓ – počtu aplikací induktivních pravidel (Definice 7.1) při sestavování formule φ.

- Báze indukce ($\ell = 0$): Pro všechny atomy, tj. výrokové proměnné, zřejmě platí, že $\mathcal{F}(A) = A$ je ekvivalentní A a $\mathcal{G}(A) = \neg A$ je ekvivalentní $\neg A$.
- V indukčním kroku předpokládejme, že a) i b) platí pro všechny formule φ délky nejvýše ℓ. Vezmeme si formuli ψ délky $\ell + 1$, která je utvořená jedním z následujících způsobů:
  - $*$ $\psi \equiv \neg\varphi$ ($\equiv$ je „definiční rovnítko“ pro formule). Podle výše uvedeného induktivního předpisu je $\mathcal{F}(\psi) = \mathcal{F}(\neg\varphi) = \mathcal{G}(\varphi)$. Podle indukčního předpokladu pak je $\mathcal{G}(\varphi)$ formule v normálním tvaru ekvivalentní $\neg\varphi = \psi$. Obdobně pro funktor $\mathcal{G}$ vyjádříme $\mathcal{G}(\psi) = \mathcal{G}(\neg\varphi) = \mathcal{F}(\varphi)$. Podle indukčního předpokladu pak je $\mathcal{F}(\varphi)$ formule v normálním tvaru ekvivalentní φ a to je dále ekvivalentní $\neg\neg\varphi = \neg\psi$ podle Tvrzení 7.6.

* $\psi \equiv (\varphi_1 \Rightarrow \varphi_2)$. Podle výše uvedeného induktivního předpisu je $\mathcal{F}(\psi) = \mathcal{F}(\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\varphi_1) \Rightarrow \mathcal{F}(\varphi_2)$. Podle indukčního předpokladu jsou $\mathcal{F}(\varphi_1)$ i $\mathcal{F}(\varphi_2)$ formule v normálním tvaru ekvivalentní φ_1 a φ_2. Potom i $\mathcal{F}(\varphi_1) \Rightarrow \mathcal{F}(\varphi_2)$ je v normálním tvaru dle definice a podle sémantiky $\Rightarrow$ je ta ekvivalentní formuli $(\varphi_1 \Rightarrow \varphi_2) = \psi$.

  Obdobně rozepíšeme $\mathcal{G}(\psi) = \mathcal{G}(\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\varphi_1) \wedge \mathcal{G}(\varphi_2)$. Jelikož $\wedge$ je pro nás jen zkratka, výraz dále rozepíšeme $\mathcal{G}(\psi) = \neg(\mathcal{F}(\varphi_1) \Rightarrow \neg\mathcal{G}(\varphi_2))$. Podle indukčního předpokladu (a dvojí negace) jsou $\mathcal{F}(\varphi_1)$ a $\neg\mathcal{G}(\varphi_2)$ po řadě ekvivalentní formulím φ_1 a φ_2. Tudíž nakonec odvodíme, že $\mathcal{G}(\psi)$ je ekvivalentní negaci formule $\varphi_1 \Rightarrow \varphi_2$, což jsme zde měli dokázat.

* $\psi \equiv (\varphi_1 \vee \varphi_2)$. Zde si musíme opět uvědomit, že spojka $\vee$ je pro nás jen zkratka, a přepsat $\psi \equiv (\neg\varphi_1 \Rightarrow \varphi_2)$. Potom podle předchozích dokázaných případů víme, že $\mathcal{F}(\psi) = \mathcal{F}(\neg\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\neg\varphi_1) \Rightarrow \mathcal{F}(\varphi_2)$ je ekvivalentní formuli $(\neg\varphi_1 \Rightarrow \varphi_2) = \psi$, což bylo třeba dokázat. Stejně tak $\mathcal{G}(\psi) = \mathcal{G}(\neg\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\neg\varphi_1) \wedge \mathcal{G}(\varphi_2)$ je podle předchozích případů důkazu ekvivalentní $(\neg\varphi_1 \wedge \neg\varphi_2) = \neg\psi$.

* $\psi \equiv (\varphi_1 \wedge \varphi_2)$ a $\psi \equiv (\varphi_1 \Leftrightarrow \varphi_2)$ už dokončíme analogicky.

□

7.4 Predikátová logika, kvantifikace

Výše popsaná výroková logika je velmi omezená faktem, že každý výrok musí být („absolutně“) vyhodnocen jako pravda nebo nepravda. Co když však chceme zpracovat tvrzení typu „den D v Brně pršelo“? Jeho pravdivostní hodnota přece závisí na tom, co dosadíme za den D, a tudíž jej nelze považovat za výrok výrokové logiky.

- *Predikátová logika* je *obecnější než logika výroková*; každá formule výrokové logiky je i formulí predikátové logiky, ale ne obráceně.

- Predikátová logika pracuje s *predikáty*. Predikáty jsou „*parametrizované výroky*“, které jsou buď pravdivé nebo nepravdivé pro každou konkrétní volbu parametrů. Výrokové proměnné lze chápat jako predikáty bez parametrů.

  `Komentář:` Pro neformální přiblížení si uvedeme několik ukázek predikátů:
  * $x > 3$ (parameterem je zde x),
  * R je ekvivalence na M (parametr R),
  * čísla x a y jsou nesoudělná (parametry x, y),
  * obecně psáno $P(x, y)$.

Definice 7.10. ***Syntaxe i sémantika predikátové logiky.***
Z predikátů lze vytvářet *predikátové formule* pomocí už známých (viz Definice 7.1) výrokových spojek a následujících tzv. *kvantifikátorů*:

- $\forall x \,.\, \varphi$ „pro každou volbu parametru x platí formule φ“,

- $\exists x \,.\, \varphi$ „existuje alespoň jedna volba parametru x, pro kterou platí φ“.

`Fakt:` Je-li každá proměnná v dané formuli kvantifikovaná (tj. formule je *uzavřená*), pak je celá formule buď pravdivá nebo nepravdivá.

Konvence 7.11. Pro lepší srozumitelnost zápisu formulí predikátové logiky se domluvíme na následujícím.

* Parametry vyskytující se v predikátech ve formuli φ někdy pro referenci vypisujeme jako „parametry“ samotné formule $\varphi(x, y, \ldots)$.
* Poukud není z kontextu jasné, co lze za daný parametr dosazovat, užívá se notace $\forall x \in M \,.\, \varphi$ a $\exists x \in M \,.\, \varphi$. (Platí, jen pokud je kvantifikovaný parametr prvkem nějaké fixní množiny!).
* Tečka za symbolem kvantifikátoru se někdy vynechává (při vhodném uzávorkování formule), nebo se používá symbol „ : “.
* Místo $\forall x_1 \,.\, \forall x_2 \,.\, \cdots \forall x_n \,.\, \varphi$ se někdy krátce píše $\forall x_1, x_2, \cdots, x_n \,.\, \varphi$. Podobně u existenčního kvantifikátoru $\exists x_1, x_2, \cdots, x_n \,.\, \varphi$.

Příklad 7.12. *Ukažme si vyjádření následujících slovních výroků v predikátové logice:*

- Každé prvočíslo větší než 2 je liché,
$$\forall n \in \mathbb{N} \,.\, \left(Pr(n) \wedge n{>}2\right) \;\Rightarrow\; Li(n).$$
- Každé číslo n, které není prvočíslem, je dělitelné nějakým číslem y kde $n \neq y$ a $y > 1$,
$$\forall n \in \mathbb{N} \,.\, \left(\neg Pr(n) \Rightarrow\; \exists y \,.\, (y|n \wedge n{\neq}y \wedge y{>}1)\right).$$
- Jsou-li R a S ekvivalence na M, je také $R \cup S$ ekvivalence na M. Zde například můžeme mít dva pohledy na toto tvrzení – v jednom bereme množinu M za pevnou
$$\forall R, S \;:\; \left(Eq_M(R) \wedge Eq_M(S)\right) \;\Rightarrow\; Eq_M(R{\cup}S),$$
kdežto ve druhém je i množina M parametrem
$$\forall M \; \forall R, S \;:\; \left(Eq(M, R) \wedge Eq(M, S)\right) \;\Rightarrow\; Eq(M, R{\cup}S).$$

□

Jak „negovat“ formule predikátové logiky?

Metoda 7.13. ***Převod predikátové formule Φ do normálního tvaru $\mathcal{F}(\Phi)$.***
Dřívější Metodu 7.8 rozšíříme o následující indukční pravidla:

$$\begin{array}{lcl@{\qquad}lcl}
\mathcal{F}(P(x_1, \ldots, x_n)) & = & P(x_1, \ldots, x_n) & \mathcal{G}(P(x_1, \ldots, x_n)) & = & \neg P(x_1, \ldots, x_n) \\
\mathcal{F}(\forall x \,.\, \varphi) & = & \forall x \,.\, \mathcal{F}(\varphi) & \mathcal{G}(\forall x \,.\, \varphi) & = & \exists x \,.\, \mathcal{G}(\varphi) \\
\mathcal{F}(\exists x \,.\, \varphi) & = & \exists x \,.\, \mathcal{F}(\varphi) & \mathcal{G}(\exists x \,.\, \varphi) & = & \forall x \,.\, \mathcal{G}(\varphi)
\end{array}$$

`Komentář:` Uvažme například formuli
$$\neg(\forall M \; \forall R, S \;:\; \left(Eq(M, R) \wedge Eq(M, S)\right) \;\Rightarrow\; Eq(M, R{\cup}S))\,.$$

Pak
$$\begin{array}{lc}
\mathcal{F}(\neg(\forall M \; \forall R, S \;:\; (Eq(M, R) \wedge Eq(M, S)) \;\Rightarrow\; Eq(M, R{\cup}S))) & = \\
\mathcal{G}(\forall M \; \forall R, S \;:\; (Eq(M, R) \wedge Eq(M, S)) \;\Rightarrow\; Eq(M, R{\cup}S)) & = \\
\exists M \; \exists R, S \;:\; \mathcal{G}((Eq(M, R) \wedge Eq(M, S)) \;\Rightarrow\; Eq(M, R{\cup}S)) & = \\
\exists M \; \exists R, S \;:\; (Eq(M, R) \;\wedge\; Eq(M, S) \;\wedge\; \neg Eq(M, R{\cup}S))\,. &
\end{array}$$

Rozšiřující studium

....

8 Dokazování vlastností algoritmů

Úvod

Po předchozí převážně matematické látce se náš výklad obrací zase k informatice a navazuje na druhý pilíř celého předmětu z Lekce 1 – algoritmy a jejich zápis. Jak jste asi již poznali, umění programovat není zdaleka jen o tom naučit se syntaxi programovacího jazyka, ale především o schopnosti vytvářet a správně formálně zapisovat algoritmy. Přitom situace, kdy programátorem zapsaný algoritmus počítá něco trochu jiného, než si programátor představuje, je určitě nejčastější programátorskou chybou, o to zákeřnější, že ji žádný „chytrý" překladač nemůže odhalit.

Proto již na počátku (seriózního) studia informatiky je dobré klást důraz na správné chápání zápisu algoritmů i na důkazy jejich vlastností a správnosti. A to je téma, kterému se tedy budeme věnovat po převážný zbytek předmětu IB000.

Cíle

Na podkladě formálního zápisu algoritmů z Oddílu 1.3 si ukážeme, jak využít předchozí nabyté matematické znalosti při dokazování vlastností a správnosti různých jednoduchých algoritmů. (Nejčastějším nástrojem nám při tom bude matematická indukce.)

8.1 O „správnosti" programů

Jak se máme přesvědčit, že je daný program „správný"?

- Co třeba ladění programů?
  Jelikož počet možných vstupních hodnot je (v principu) neohraničený, nelze otestovat všechna možná vstupní data.
- Situace je zvláště komplikovaná v případě paralelních, randomizovaných, interaktivních a nekončících programů (operační systémy, systémy řízení provozu apod.). Takové systémy mají nedeterministické chování a opakované experimenty tudíž vedou k různým výsledkům. (Nelze je rozumně ladit, respektive ladění poskytne jen velmi nedostatečnou záruku správného chování za jiných okolností.)
- V některých případech je však třeba mít naprostou jistotu, že program funguje tak jak má, případně že splňuje základní bezpečnostní požadavky.
  Narůstající složitost programových systémů a zvýšené požadavky na jejich bezpečnost si vynucují vývoj „spolehlivých" formálních verifikačních metod.

8.2 Jednoduché indukční dokazování

Pro svěžení znalostí se nejprve podívejte zpět na naše konvence formálního zápisu algoritmů do Oddílu 1.3. Náš výklad začneme s několika skutečně jednoduchými algoritmy, jejichž jediný účel je v demonstraci využití matematické indukce pro dokazování vlastností.

Příklad 8.1. *Zjistěte, kolik znaků* `'x'` *v závislosti na celočíselné hodnotě* n *vstupního parametru* `n` *vypíše následující algoritmus.*

Algoritmus 8.2.

```
for  i ← 1,2,3,...,n-1,n  do
    for  j ← 1,2,3,...,i-1,i  do
        vytiskni  'x';
    done
done
```

Nejprve si uvědomíme, že druhý (vnořený) cyklus vždy vytiskne celkem i znaků `'x'`. Proto iterací prvního cyklu (nejspíše) dostaneme postupně $1 + 2 + \ldots + n$ znaků `'x'` na výstupu, což již víme (Příklad 2.7), že je celkem $\frac{1}{2}n(n+1)$. Budeme tedy dokazovat následující tvrzení:

Věta. Pro každé přirozené n Algoritmus 12.3 vypíše právě $\frac{1}{2}n(n+1)$ znaků `'x'`.

`Důkaz`: Postupujeme indukcí podle n. Báze pro $n = 0$ je zřejmá, neprovede se ani jedna iterace cyklu a tudíž bude vytištěno 0 znaků `'x'`, což máme dokázat.

Nechť tedy tvrzení platí pro jakékoliv n_0 a položme $n = n_0 + 1$. Prvních n_0 iterací vnějšího cyklu podle indukčního předpokladu vypíše (ve vnitřním cyklu) celkem $\frac{1}{2}n_0(n_0+1)$ znaků `'x'`. Pak již následuje jen jedna poslední iterace vnějšího cyklu s `i` $\leftarrow$ `n=n`$_0$`+1` a v ní se vnitřní cyklus `j` $\leftarrow$ `1,2,...,i=n` iteruje celkem $n = n_0 + 1$ -krát. Celkem tedy bude vytištěn tento počet znaků `'x'`:

$$\frac{1}{2}n_0(n_0+1) + n_0 + 1 = \frac{1}{2}(n_0+1+1)(n_0+1) = \frac{1}{2}n(n+1)$$

Důkaz indukčního kroku je hotov. □

Příklad 8.3. *Zjistěte, kolik znaků* `'z'` *v závislosti na celočíselné hodnotě* n *vstupního parametru* `n` *vypíše následující algoritmus.*

Algoritmus 8.4.

```
st  ← "z";
for  i ← 1,2,3,...,n-1,n  do
    vytiskni řetězec  st;
    st  ←  st+st ;    (zřetězení dvou kopií st za sebou)
done
```

Zkusíme-li si výpočet simulovat pro $n = 0, 1, 2, 3, 4\ldots$, postupně dostaneme počty `'z'` jako $0, 1, 3, 7, 15\ldots$. Na základě toho již není obtížné „uhodnout“, že počet `'z'` bude (asi) obecně určen vztahem $2^n - 1$. Toto je však třeba dokázat!

`Komentář:` Jak záhy zjistíme, matematická indukce na naše tvrzení přímo „nezabírá“, ale mnohem lépe se nám povede s následujícím přirozeným zesílením dokazovaného tvrzení:

Věta. Pro každé přirozené n Algoritmus 8.4 vypíše právě $2^n - 1$ znaků `'z'` a proměnná `st` bude na konci výpočtu obsahovat řetězec 2^n znaků `'z'`.

`Důkaz`: Postupujeme indukcí podle n. Báze pro $n = 0$ je zřejmá, neprovede se ani jedna iterace cyklu a tudíž bude vytištěno 0 znaků `'z'`, což máme dokázat.

Nechť tedy tvrzení platí pro jakékoliv n_0 a položme $n = n_0 + 1$. Podle indukčního předpokladu po prvních n_0 iteracích bude vytištěno $2^{n_0} - 1$ znaků `'z'` a proměnná `st` bude obsahovat řetězec 2^{n_0} znaků `'z'`. V poslední iteraci cyklu (pro `i` $\leftarrow$ `n=n`$_0$`+1`) vytiskneme dalších 2^{n_0} znaků `'z'` (z proměnné `st`) a dále řetězec `st` „zdvojnásobíme“. Proto po n iteracích bude vytištěno celkem $2^{n_0}-1+2^{n_0}=2^{n_0+1}-1=2^n-1$ znaků `'z'` a v `st` bude uloženo $2\cdot 2^{n_0}=2^n$ znaků `'z'`. □

8.3 Algoritmy pro relace

Relace jsou velice vhodnou strukturou pro algoritmické zpracování. Proto si uvedeme ukázky tří „abstraktních“ algoritmů pro práci s relacemi, včetně jejich důkazů. (Pilní studenti si zajisté dokážou navrhnout sami i další podobné algoritmy.)

Algoritmus 8.5. ***Symetrický uzávěr.***
Pro danou relaci R na n-prvkové množině $A=\{a_1,a_2,\dots,a_n\}$ vytvoříme její symetrický uzávěr $\overleftrightarrow{R}$ takto:

```
R⃡ ← R ;
for  i ← 1,2,...,n-1,n  do
    for  j ← 1,2,...,n-1,n  do
        if  (a_i,a_j) ∈ R ∧ (a_j,a_i) ∉ R  then R⃡ ← R⃡ ∪ {(a_j,a_i)} ;
    done
done
```

`Důkaz`: Zde není důkaz vůbec obtížný. Relace $\overleftrightarrow{R}$ je zřejmě symetrická, neboť (vnitřní) tělo cyklu pro všechny dvojice $(a_i,a_j)\in R$ přidá i (a_j,a_i). Z druhé strany všechny dvojice „přidané“ v $\overleftrightarrow{R}\setminus R$ musí být obsaženy podle definice symetrické relace, takže $\overleftrightarrow{R}$ je skutečně symetrickým uzávěrem podle definice uzávěru relace. □

`Poznámka`: Všimněte si, že jsme tento algoritmus zapsali *abstraktně* – vůbec jsme nekonkretizovali datové typy a struktury pro zápis relací. (Náš algoritmus jako takový tyto konkretizace nepotřebuje.) Na jednu stranu je to výhodné, neboť algoritmus můžeme pak použít na libovolně implementovan relace. V neposlední řadě je takový abstraktní zápis i přehlednější a snadněji pochopitelný.

Na druhou stranu však programátor sám musí teď zvolit vhodnou implementaci relace, například jako dvourozměrné pole `R`, kde `R[i,j]=0` právě když $(a_i,a_j)\notin R$. Následně je třeba zdůvodnit i *správnost zvolené implementace*(!).

Algoritmus 8.6. ***Tranzitivní uzávěr.***
Pro danou relaci R na n-prvkové množině $A=\{a_1,a_2,\dots,a_n\}$ vytvoříme její tranzitivní uzávěr R^+ takto:

```
R⁺ ← R ;
for  k ← 1,2,...,n-1,n  do
    for i ← 1,2,...,n-1,n do  for j ← 1,2,...,n-1,n do
        if  (a_i,a_k) ∈ R⁺ ∧ (a_k,a_j) ∈ R⁺  then
            if  (a_i,a_j) ∉ R⁺  then R⁺ ← R⁺ ∪ {(a_i,a_j)} ;
        fi
    done
done
```

Jak by se dala dokázat správnost popsaného algoritmu? Přímá aplikace indukce podle n nevypadá přínosně... (Zkuste si sami!) Nejkratší cesta k cíli vede použitím indukce (podle proměnné k vnějšího cyklu) na vhodně zesíleném tvrzení. Pro jeho formulaci si definujeme, že relace S na A je *k-částečně tranzitivní*, pokud pro libovolná i, j a pro $\ell \leq k$ platí, že z $(a_i, a_\ell), (a_\ell, a_j) \in S$ vyplývá $(a_i, a_j) \in S$. (Všimněte si, že pro $k = 0$ tato definice neříká nic a pro $k = n$ znamená běžnou tranzitivní relaci.)

> *Věta.* Po každých $k \geq 0$ iteracích vnějšího cyklu Algoritmu 8.6 aktuální hodnota relace R^+ udává k-částečně tranzitivní uzávěr relace R na A.

Důkaz: Báze indukce pro $k = 0$ jasně platí, neboť věta v tom případě nic neříká.

Předpokládejme nyní, že tvrzení platí pro nějaké $k_0 \geq 0$ a dokažme jej i pro $k = k_0+1$. Zřejmě stačí uvažovat případ $k_0 < n$. Každá dvojice (a_i, a_j) přidaná do R^+ uvnitř cyklu musí náležet do k-částečně tranzitivního uzávěru podle definice. Zbývá zdůvodnit, proč každá dvojice (a_i, a_j) náležející do k-částečně tranzitivního uzávěru, ale ne do k_0-částečně tranzitivního uzávěru, bude do R^+ v k-té iteraci přidána.

Není těžké ověřit, že (a_i, a_j) náleží do k-částečně tranzitivního uzávěru, právě když v relaci R nalezneme takovou cestu „po šipkách" z a_i do a_j, která přechází pouze přes prvky a_ℓ kde $\ell \leq k$. V naší situaci vyplývá, že taková cesta musí použít i prvek a_k (jen jednou!), a proto (a_i, a_k) i (a_k, a_j) náleží do k_0-částečně tranzitivního uzávěru R. V k-té iteraci tudíž bude příslušná `if` podmínka splněná a (a_i, a_j) bude přidána do R^+. $\square$

Dokazování konečnosti algoritmu

Komentář: Všimněte si, že jsme se zatím v důkazech vůbec nezamýšleli nad tím, zda náš algoritmus vůbec skončí. (To jistě není samozřejmé a důkaz konečnosti je nutno v obecnosti podávat!) Prozatím jsme však ukazovali algoritmy využívající jen `for` cykly, přitom podle naší konvence obsahuje `for` cyklus předem danou konečnou množinu hodnot pro řídící proměnnou, neboli náš `for` cyklus vždy musí skončit. Ale už v příštím algoritmu využijeme `while` cyklus, u kterého vůbec není jasné kdy a jestli skončí, a tudíž bude potřebný i důkaz konečnosti.

Metoda 8.7. ***Důkaz konečnosti.***
Máme-li za úkol dokázat, že algoritmus skončí, postupujeme nejlépe následovně:

* *Sledujeme zvolený celočíselný a zdola ohraničený parametr algoritmu (třeba přirozené číslo) a dokážeme, že se jeho hodnota v průběhu algoritmu neustále ostře zmenšuje.*
* *Případně předchozí přístup rozšíříme na zvolenou k-tici přirozených parametrů a dokážeme, že se jejich hodnoty v průběhu algoritmu lexikograficky ostře zmenšují.*

Pozor, naše „parametry" vůbec nemusejí být proměnnými v programu.

Algoritmus 8.8. ***Cykly permutace.***
Pro danou permutaci π na n-prvkové neprázdné množině $A = \{1, 2, \dots, n\}$ vypíšeme její cykly (viz Oddíl 6.4) takto:

```
U ← {1,2,...,n};
while U≠∅ do
    x ← min(U);     (nejmenší prvek množiny)
    začínáme výpis cyklu '⟨';
    while x∈U do
        vytiskneme x;
```

```
        U ← U\{x};    x ← π(x);
    done
    ukončíme výpis cyklu ’⟩’;
done
```

Jak dokážeme správnost tohoto algoritmu? Opět platí, že přímá aplikace indukce podle n nepřinese nic podstatného. Důkaz si tentokrát rozdělíme na dvě části (podle dvou `while` cyklů). Všimněte se navíc, že tentokrát je nezbytnou součástí důkazu správnosti algoritmu i důkaz, že oba `while` cykly vždy skončí.

> *Věta.* Za předpokladu, že vnitřní `while` cyklus pro jakoukoliv počáteční volbu `x` skončí, vypíše cyklus permutace π obsahující x a odebere všechny prvky tohoto cyklu z množiny `U`, Algoritmus 8.8 vždy skončí se správným výsledkem.

Důkaz: Postupujeme indukcí podle počtu cyklů v permutaci π. Jediný cyklus v π (báze indukce) je vypsán dle předpokladu věty a množina U zůstane prázdná, tudíž vnější `while` cyklus skončí po první iteraci a výsledek je správný.

Podle Věty 6.5 se každá permutace dá zapsat jako složení disjunktních cyklů. Nechť π je tedy složena z $\ell > 1$ cyklů. Po první iteraci `while` cyklu zbude v restrikci permutace π na množinu U celkem $\ell - 1$ cyklů. Podle indukčního předpokladu pak tyto zbylé cykly budou správně vypsány a algoritmus skončí. □

Komentář: Vidíte, že v tomto důkaze indukcí je indukční krok zcela triviální a důležitý je zde především základ indukce?

> *Věta.* Pokud π je permutace, tak vnitřní `while` cyklus vždy skončí a nalezne v π cyklus obsahující libovolný počáteční prvek $x \in U$. Navíc všechny prvky nalezeného cyklu odebere z množiny U.

Důkaz: Zde přímo zopakujeme argument důkazu Věty 6.5: Vezmeme libovolný prvek $x = x_1 \in A$ a iterujeme zobrazení $x_{i+1} = \pi(x_i)$ pro $i = 1, 2 \ldots$, až dojde ke zopakování prvku $x_k = x_j$ pro $k > j \geq 1$. (To musí nastat, neboť A je konečná.) Jelikož prvek x_j byl již odebrán z U, v kroku $x = x_k$ dojde k ukončení našeho `while` cyklu. Nadto je π prostá, a proto nemůže nastat $x_k = x_j = \pi(x_{j-1})$ pro $j > 1$. Takto byl nalezen a odebrán z U cyklus $\langle a_1, \ldots, a_{k-1} \rangle$ a důkaz je hotov. □

8.4 Zajímavé algoritmy aritmetiky

Pro další ukázky důkazových technik pro algoritmy se podíváme na některé méně známé krátké algoritmy z oblasti aritmetiky. Například celočíselné umocňování na velmi vysoké exponenty je základem známé RSA šifry:

Algoritmus 8.9. ***Binární postup umocňování.***
Pro daná čísla a, b vypočteme jejich celočíselnou mocninu (omezenou na zbytkové třídy modulo m kvůli prevenci přetečení rozsahu celých čísel v počítači), tj. $c = a^b \bmod m$.

```
c ← 1;
while  b > 0  do
    if  b mod 2 > 0  then c ← (c·a) mod m;
    b ← ⌊b/2⌋;   a ← (a·a) mod m;
done
výsledek c;
```

Zde použijeme k důkazu správnosti algoritmu indukci podle délky ℓ binárního zápisu čísla b.

> *Věta.* Algoritmus 8.9 skončí a vždy správně vypočte hodnotu mocniny $c = a^b \bmod m$.

`Důkaz`: Báze indukce je pro $\ell = 1$, kdy $b = 0$ nebo $b = 1$. Přitom pro $b = 0$ se cyklus vůbec nevykoná a výsledek je $c = 1$. Pro $b = 1$ se vykoná jen jedna iterace cyklu a výsledek je $c = a \bmod m$.

Nechť tvrzení platí pro $\ell_0 \geq 1$ a uvažme $\ell = \ell_0 + 1$. Pak zřejmě $b \geq 2$ a vykonají se alespoň dvě iterace cyklu. Po první iteraci bude $a' = a^2$, $b' = \lfloor b/2 \rfloor$ a $c' = (a^{b \bmod 2}) \bmod m$. Tudíž délka binárního zápisu b' bude jen ℓ_0 a podle indukčního předpokladu zbylé iterace algoritmu skončí s výsledkem

$$c = c' \cdot {a'}^{b'} \bmod m = (a^{b \bmod 2} \cdot a^{2\lfloor b/2 \rfloor}) \bmod m = a^b \bmod m\,.$$

□

Na závěr lekce si ukážeme jeden netradiční krátký algoritmus a jeho analýzu a důkaz ponecháme zde otevřené. Dokážete popsat, na čem je algoritmus založen?

Algoritmus 8.10. ***Celočíselná odmocnina.***
Pro dané číslo x vypočteme dolní celou část jeho odmocniny $r = \lfloor \sqrt{x} \rfloor$.

```
p ← x;   r ← 0;
while  p > 0  do
    while  (r + p)^2 < x   do r  ←  r+p ;
    p  ←  ⌊p/2⌋ ;
done
výsledek r ;
```

`Poznámka`: Zamysleli jste se, jaký mají algoritmy v tomto oddíle vlastně význam? Vždyť stejné úlohy jistě sami vyřešíte každý jednou jednoduchou `for` smyčkou. Podívejte se však (alespoň velmi zhruba) na počet kroků, které zde uvedené algoritmy potřebují vykonat k získání výsledku, a srovnejte si to s počty kroků oněch „jednoduchých“ algoritmů.

Pro skutečně velká vstupní čísla zjistíte propastný rozdíl – s takovým „jednoduchým“ algoritmem, třeba '`for i ← 1,...b do c ← c·a mod m done`', se pro obrovské hodnoty b výsledku nikdy nedočkáte, kdežto Algoritmus 8.9 stále poběží velmi rychle. (Spočítáte, jak rychle?)

Rozšiřující studium

....

9 Jednoduchý deklarativní jazyk

Úvod

Pokračujeme dále v tématu předchozí lekce, tj. budeme se zabývat matematickým dokazováním vlastností a správnosti algoritmů. Třebaže mnohým mohla přijít už Lekce 8 více než dost formální, není tomu úplně tak; nyní si ukážeme (ještě) přesnější přístup založený na myšlenkách funkcionálního programování.

Cíle

Naším hlavním cílem je podání matematicky zcela přesné definice jednoduchého deklarativního „programovacího“ jazyka pro potřeby formálního dokazování příslušně zapsaných algoritmů. (Na tomto jazyce pak v příští lekci postavíme přehled důkazových technik pro algoritmy.)

O „správnosti“ programů, podruhé

Vraťme se k otázce, jak se máme přesvědčit, že program funguje „správně“?

* V některých případech, jak už jsme zmínili, je třeba mít naprostou jistotu, že program funguje tak jak má, například v řídících systémech, na nichž závisí lidské životy. V takovém případě je jedinou „dostatečně spolehlivou“ možností podat formální matematický důkaz chování algoritmu.
* A co tedy důkazy vlastností symbolicky zapsaných (procedurálních) algoritmů z Lekce 8? Všimli jste si, co v nich bylo problematickým bodem? Náš procedurální zápis algoritmu totiž přesně nedefinuje, co je to „elementární krok“ výpočtu – to je sice většinou docela zřejmé, někdy však může hlavní problém nastat právě zde. Sice by bylo možné použít k definici některý z přesných teoretických modelů výpočtu jako je *Turingův stroj* (nebo třeba i některý vhodný z reálných programovacích jazyků), avšak pak by se formální důkazy staly velmi složitými.
* Vhodnějším řešením (pro potřeby formálního dokazování) se jeví příklon k „funkcionálnímu“ zápisu algoritmů pomocí matematicky zcela přesných *deklarací*.

9.1 Popis jednoduchého deklarativního jazyka

Z těchto výše popsaných důvodů se nyní soustředíme na podání matematicky zcela přesné definice jednoduchého deklarativního „programovacího“ jazyka. Začneme jeho syntaxí a poté přejdeme k jeho sémantice – tedy k formalizaci takto zapsaných pojmů „výpočetního kroku“ a „výpočtu“.

Definice 9.1. ***Deklarativní programovací jazyk*** (pro přednášky IB000).

* Nechť $\mathit{Var} = \{x, y, z, \ldots\}$ je spočetná množina *proměnných*.
* Nechť $\mathit{Num} = \{\mathbf{0}, \mathbf{1}, \ldots \mathbf{52}, \ldots \mathbf{397}, \ldots\}$ je množina všech dekadických zápisů *přirozených* čísel.
* Nechť $\mathit{Fun} = \{f, g, h, \ldots\}$ je spočetná množina *funkčních symbolů*. Ke každému $f \in \mathit{Fun}$ je přiřazeno číslo $a \in \mathbb{N}$, které nazýváme *arita* f. Dále předpokládáme, že pro každé $a \in \mathbb{N}$ existuje nekonečně mnoho $f \in \mathit{Fun}$ s aritou a.
* Množina ***výrazů*** Exp je (*induktivně*) definována následující abstraktní syntaktickou rovnicí:

$$\begin{array}{rcl} E & ::= & x \mid \mathbf{n} \\ & \mid & E_1 + E_2 \mid E_1 - E_2 \mid E_1 * E_2 \mid E_1 \div E_2 \mid (\,E_1\,) \\ & \mid & f(E_1, \cdots, E_a) \\ & \mid & \mathbf{if}\ E_1\ \mathbf{then}\ E_2\ \mathbf{else}\ E_3 \end{array}$$

V uvedené rovnici je $x \in \mathit{Var}$, $\mathbf{n} \in \mathit{Num}$, $E_i \in \mathit{Exp}$ jsou výrazy, $f \in \mathit{Fun}$ a $a \in \mathbb{N}$ je arita funkce f.

Poznámka: Takováto specifikace syntaxe je *abstraktní* v tom smyslu, že se nezabývá tím, jak výrazy jednoznačně zapsat do řádku jako posloupnost symbolů. Je na nás, abychom napsali dostatečně mnoho závorek a případně stanovili prioritu operátorů tak, aby bylo zcela jasné, jak daný výraz podle uvedené rovnice vznikl. (Ve smyslu Lekce 6 tato induktivní definice není jednoznačná. To nám však nebude v Definici 9.2 vadit.)

Komentář: Pro lepší pochopení uvádíme několik příkladů výrazů (*Exp*) z definice.

- $\mathbf{254}$
- $\mathbf{2} + \mathbf{3} * \mathbf{4}$
- $f(\mathbf{2}) \div g(\mathbf{5})$
- $f(\mathbf{2} + x, g(y, \mathbf{3} * y))$
- $\textbf{if } x - \mathbf{1} \textbf{ then } (\mathbf{2} + f(y)) \textbf{ else } g(x, x)$

(Vyhodnocení podmínky v „if" testuje nenulovost argumentu.)

Definice: *Deklarace* (v jazyce Definice 9.1) je konečný systém rovnic tvaru

$$\begin{array}{rcl} f_1(x_1, \cdots, x_{a_1}) & = & E_1 \\ \vdots & & \vdots \\ f_n(x_1, \cdots, x_{a_n}) & = & E_n \end{array},$$

kde pro každé $1 \leq i \leq n$ platí, že $f_i \in Fun$ je funkce arity a_i, že $x_1, \ldots, x_{a_i} \in Var$ jsou proměnné a E_i je výraz, v němž se mohou vyskytovat pouze proměnné $x_1, \ldots, x_{a_i}$ a funkční symboly $f_1, \ldots, f_n$.

Komentář: Opět uvádíme pro osvětlení několik příkladů deklarací z naší definice.

- $f(x) \;=\; \textbf{if } x \textbf{ then } x * f(x - \mathbf{1}) \textbf{ else } \mathbf{1}$
- $\begin{array}{rcl} f(x) & = & g(x - \mathbf{1}, x) \\ g(x, y) & = & \textbf{if } x \textbf{ then } f(y) \textbf{ else } \mathbf{3} \end{array}$

  (Jak uvidíme formálně později, konvencí našich výpočtů je neužívat záporná čísla, místo toho $0 - 1$ *„="* 0.*)*
- $g(x, y) \;=\; \textbf{if } x - y \textbf{ then } x \textbf{ else } y$
- $f(x) \;=\; f(x)$

  (Nezapisuje toto náhodou „nekonečnou smyčku"?)

Deklarace tedy udává „soubor pravidel", podle kterých vyhodnocujeme (Definice 9.2) daný výraz. Jinak také lze deklaraci chápat jako zobecnění *rekurentních definic funkcí.*

9.2 Formalizace pojmu „výpočet"

Za výpočet (nad Δ) budeme považovat posloupnost úprav výrazů, které jsou „postaveny" na naší uvažované deklaraci Δ. To je formálně podchyceno následujícíma dvěma definicema.

Definice: Buď Δ deklarace. Symbolem *Exp*(Δ) označíme množinu všech výrazů E, které splňují zároveň tyto dvě podmínky:

– E neobsahuje žádnou proměnnou.

– Jestliže E obsahuje funkční symbol f, pak f byl v Δ deklarován (tj. na levé straně některé rovnice Δ).

`Fakt`: Množinu $\mathit{Exp}(\Delta)$ lze definovat také induktivně:

$$\begin{array}{rcl} E & ::= & \mathbf{n} \mid (E_1) \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 * E_2 \mid E_1 \div E_2 \\ & \mid & f(E_1, \cdots, E_a) \mid \mathbf{if}\ E_1\ \mathbf{then}\ E_2\ \mathbf{else}\ E_3 \end{array}$$

V uvedené zápise je $\mathbf{n} \in \mathit{Num}$, $f \in \mathit{Fun}$ je funkční symbol deklarovaný v Δ a $a \in \mathbb{N}$ je arita tohoto f.

Definice 9.2. ***Výpočet a krok výpočtu*** v našem deklarativním jazyce.
Funkci „*krok výpočtu*“ $\mapsto\ : \mathit{Exp}(\Delta) \to \mathit{Exp}(\Delta)$ definujeme induktivně takto; místo $\mapsto(E) = F$ budeme psát $\boldsymbol{E \mapsto F}$.

- $\mathbf{n} \mapsto \mathbf{n}$ pro každé $\mathbf{n} \in \mathit{Num}$.
- Pro $\boldsymbol{E \equiv (E_1)}$ definujeme krok výpočtu takto:
  - ∗ Jestliže $E_1 \mapsto F \in \mathit{Num}$, pak $(E_1) \mapsto F$.
  - ∗ Jestliže $E_1 \mapsto F \notin \mathit{Num}$, pak $(E_1) \mapsto (F)$.
- Pro $\boldsymbol{E \equiv E_1 + E_2}$ definujeme krok výpočtu takto:
  - ∗ Jestliže $E_1, E_2 \in \mathit{Num}$, pak $E_1 + E_2 \mapsto \mathbf{z}$, kde $\mathbf{z}$ je dekadický zápis čísla $E_1 + E_2$.
  - ∗ Jestliže $E_1 \notin \mathit{Num}$, pak $E_1 + E_2 \mapsto F + E_2$, kde $E_1 \mapsto F$.
  - ∗ Jestliže $E_1 \in \mathit{Num}$ a $E_2 \notin \mathit{Num}$, pak $E_1 + E_2 \mapsto E_1 + F$, kde $E_2 \mapsto F$.
- Pro $\boldsymbol{E \equiv E_1 - E_2}$ definujeme krok výpočtu takto:
  - ∗ Jestliže $E_1, E_2 \in \mathit{Num}$, pak $E_1 - E_2 \mapsto \mathbf{z}$, kde $\mathbf{z}$ je dekadický zápis čísla $\max\{0, E_1 - E_2\}$. (Pozor na nezápornost výsledku odčítání!)
  - ∗ Jestliže $E_1 \notin \mathit{Num}$, pak $E_1 - E_2 \mapsto F - E_2$, kde $E_1 \mapsto F$.
  - ∗ Jestliže $E_1 \in \mathit{Num}$ a $E_2 \notin \mathit{Num}$, pak $E_1 - E_2 \mapsto E_1 - F$, kde $E_2 \mapsto F$.
- Pro $\boldsymbol{E \equiv E_1 * E_2}$ definujeme krok výpočtu takto:
  - ∗ Jestliže $E_1, E_2 \in \mathit{Num}$, pak $E_1 * E_2 \mapsto \mathbf{z}$, kde $\mathbf{z}$ je dekadický zápis čísla $E_1 * E_2$.
  - ∗ Jestliže $E_1 \notin \mathit{Num}$, pak $E_1 * E_2 \mapsto F * E_2$, kde $E_1 \mapsto F$.
  - ∗ Jestliže $E_1 \in \mathit{Num}$ a $E_2 \notin \mathit{Num}$, pak $E_1 * E_2 \mapsto E_1 * F$, kde $E_2 \mapsto F$.
- Pro $\boldsymbol{E \equiv E_1 \div E_2}$ definujeme krok výpočtu takto:
  - ∗ Jestliže $E_1, E_2 \in \mathit{Num}$, pak $E_1 \div E_2 \mapsto \mathbf{z}$, kde $\mathbf{z}$ je dekadický zápis (celé části) čísla $\lfloor E_1/E_2 \rfloor$. Pokud $E_2 \equiv \mathbf{0}$, je $\mathbf{z} \equiv \mathbf{0}$ (dělení nulou!).
  - ∗ Jestliže $E_1 \notin \mathit{Num}$, pak $E_1 \div E_2 \mapsto F \div E_2$, kde $E_1 \mapsto F$.
  - ∗ Jestliže $E_1 \in \mathit{Num}$ a $E_2 \notin \mathit{Num}$, pak $E_1 \div E_2 \mapsto E_1 \div F$, kde $E_2 \mapsto F$.
- Pro $\boldsymbol{E \equiv \mathbf{if}\ E_1\ \mathbf{then}\ E_2\ \mathbf{else}\ E_3}$ definujeme krok výpočtu takto:
  - ∗ Jestliže $E_1 \in \mathit{Num}$ a $E_1 \equiv \mathbf{0}$, pak $\mathbf{if}\ E_1\ \mathbf{then}\ E_2\ \mathbf{else}\ E_3 \mapsto E_3$.
  - ∗ Jestliže $E_1 \in \mathit{Num}$ a $E_1 \not\equiv \mathbf{0}$, pak $\mathbf{if}\ E_1\ \mathbf{then}\ E_2\ \mathbf{else}\ E_3 \mapsto E_2$.

- $*$ Jestliže $E_1 \not\in Num$, pak **if** E_1 **then** E_2 **else** $E_3 \mapsto$**if** F **then** E_2 **else** E_3, kde $E_1 \mapsto F$.

- Pro $\boldsymbol{E \equiv f(E_1, \cdots, E_k)}$ definujeme krok výpočtu takto:
  - $*$ Jestliže $E_1, \cdots, E_k \in Num$, pak $f(E_1, \cdots, E_k) \mapsto (E_f(x_1 \restriction E_1, \cdots, x_k \restriction E_k))$. (V tomto formálním zápise se jedná o prosté „textové“ dosazení hodnot E_i za proměnné x_i v deklaraci E_f funkce f v Δ.)
  - $*$ Jinak $f(E_1, \cdots, E_k) \mapsto f(E_1, \cdots, E_{i-1}, F, E_{i+1}, \cdots, E_k)$, kde i je nejmenší index pro který platí $E_i \not\in Num$ a $E_i \mapsto F$.

V zápise jednotlivých bodů vždy platí, že $E_1, E_2, \ldots \in Exp(\Delta)$. Znak $\equiv$ zde znamená „textovou“ rovnost na množině výrazů *Exp*. Při nejednoznačnosti vždy aplikujeme první použitelné pravidlo naší definice.

Reflexivní a tranzitivní uzávěr relace $\mapsto$ značíme $\mapsto^*$ („*výpočet*“).

Tato rozsáhlá a důležitá definice si zaslouží několik podstatných připomínek. Za prvé si dobře povšimněte některých „aritmetických“ aspektů výpočtu.

– Výsledek odečítání je vždy nezáporný, neboli všechna záporná čísla jsou nahrazena nulou.

– Výsledek dělení je vždy celočíselný, počítáme jeho dolní celou část.

– Dělení nulou je definováno (není chybou), výsledkem je opět nula.

Další připomínka se týká pořadí vyhodnocování ve výrazu — to je přesně dáno pořadím pravidel v Definici 9.2, neboli vždy aplikujeme první pravidlo, které aktuálně lze použít na výraz E, a to na prvním možném místě. Mimo jiné je takto „definována“ nejvyšší priorita vyhodnocení uzávorkovaného výrazu.

Uvědomte si dobře, že definice výpočetního kroku $\mapsto$ je (poněkud skrytě) rekurzivní. Třeba krok $(\mathbf{2} * \mathbf{1}) \mapsto \mathbf{2}$ je ve skutečnosti jediným krokem, přestože „vyvolá“ použití dvou pravidel v sobě – vyhodnocení součinu i odstranění závorek.

Ještě si uveďme, že naše definice připouští jistý nedeterminismus (poznámka jen pro čtenáře, kteří o nedeterminismu už slyšeli): Je možné mít v deklaraci Δ zadaných více rovnic pro tutéž funkci $f()$, pak se však z $\mapsto$ stává pouhá relace. My se touto možností nebudeme zabývat.

9.3 Příklady výpočtů a důkazů

Příklad 9.3. *Ukážeme si několik ilustrativních „výpočtů“ nad různými deklaracemi.*

Uvažme deklaraci $f(x) = \textbf{if } x \textbf{ then } x * f(x - \mathbf{1}) \textbf{ else } \mathbf{1}$. Pak třeba $f(\mathbf{3}) \mapsto^* \mathbf{6}$, neboť

$$\begin{array}{llllll}
f(\mathbf{3}) & \mapsto & \textbf{if } \mathbf{3} \textbf{ then } \mathbf{3} * f(\mathbf{3}-\mathbf{1}) \textbf{ else } \mathbf{1} & \mapsto & \mathbf{3} * f(\mathbf{3}-\mathbf{1}) & \mapsto \\
\mathbf{3} * f(\mathbf{2}) & \mapsto & \mathbf{3} * (\textbf{if } \mathbf{2} \textbf{ then } \mathbf{2} * f(\mathbf{2}-\mathbf{1}) \textbf{ else } \mathbf{1}) & \mapsto & \mathbf{3} * (\mathbf{2} * f(\mathbf{2}-\mathbf{1})) & \mapsto \\
\mathbf{3} * (\mathbf{2} * f(\mathbf{1})) & \mapsto & \mathbf{3} * (\mathbf{2} * (\textbf{if } \mathbf{1} \textbf{ then } \mathbf{1} * f(\mathbf{1}-\mathbf{1}) \textbf{ else } \mathbf{1})) & \mapsto & \mathbf{3}*(\mathbf{2}*(\mathbf{1}*f(\mathbf{1}-\mathbf{1}))) & \mapsto \\
\mathbf{3}*(\mathbf{2}*(\mathbf{1}*f(\mathbf{0}))) & \mapsto & \mathbf{3}*(\mathbf{2}*(\mathbf{1}*(\textbf{if } \mathbf{0} \textbf{ then } \mathbf{0} * f(\mathbf{0}-\mathbf{1}) \textbf{ else } \mathbf{1}))) & \mapsto & \mathbf{3} * (\mathbf{2} * (\mathbf{1} * \mathbf{1})) & \mapsto \\
\mathbf{3} * (\mathbf{2} * \mathbf{1}) & \mapsto & \mathbf{3} * \mathbf{2} & \mapsto & \mathbf{6}\,. &
\end{array}$$

Uvažme deklaraci $f(x) = g(x - \mathbf{1}, x)$ a $g(x, y) = \textbf{if } x \textbf{ then } f(y) \textbf{ else } \mathbf{3}$. Pak například $f(\mathbf{3}) \mapsto^* f(\mathbf{3})$, neboť

$$f(\mathbf{3}) \mapsto g(\mathbf{3}-\mathbf{1}, \mathbf{3}) \mapsto g(\mathbf{2}, \mathbf{3}) \mapsto \textbf{if } \mathbf{2} \textbf{ then } f(\mathbf{3}) \textbf{ else } \mathbf{3} \mapsto f(\mathbf{3})\,.$$

Uvažme deklaraci $f(x) = f(x)$. Pak pro každé $\mathbf{n} \in \mathit{Num}$ platí $f(\mathbf{n}) \mapsto f(\mathbf{n})$ a podobně $f(f(\mathbf{n})) \mapsto f(f(\mathbf{n}))$. Ale $f(f(\mathbf{2}+\mathbf{3})) \mapsto f(f(\mathbf{5})) \mapsto f(f(\mathbf{5}))$. $\square$

Důkaz správnosti programu

Celá naše formalizace deklarativního jazyka směřuje k přesným matematickým důkazům algoritmů, takže si takové hned názorně ukážeme.

Příklad 9.4. *Pro ukázku uvažme deklaraci Δ obsahující pouze rovnici*

$$f(x) = \textbf{\textit{if}}\ x\ \textbf{\textit{then}}\ x * f(x-\mathbf{1})\ \textbf{\textit{else}}\ \mathbf{1}\,.$$

Věta. Pro každé $n \in \mathbb{N}$ platí $f(\mathbf{n}) \mapsto^* \mathbf{m}$, kde $\mathbf{m} \equiv n!$.

`Důkaz` povedeme indukcí podle n:

- *Báze* $n = 0$. Platí $f(\mathbf{0}) \mapsto \textbf{if } \mathbf{0} \textbf{ then } \mathbf{0} * f(\mathbf{0}-\mathbf{1}) \textbf{ else } \mathbf{1} \mapsto \mathbf{1}$ a platí $0! = 1$.
- *Indukční krok*. Nechť $n+1 \equiv \mathbf{k}$. Pak
$$f(\mathbf{k}) \mapsto \textbf{if k then k} * f(\mathbf{k}-\mathbf{1}) \textbf{ else } \mathbf{1} \mapsto \mathbf{k} * f(\mathbf{k}-\mathbf{1}) \mapsto \mathbf{k} * f(\mathbf{w})\,,$$
kde $\mathbf{w} \equiv k-1 = n$. Podle I.P. platí $f(\mathbf{w}) \mapsto^* \mathbf{u}$, kde $\mathbf{u} \equiv n!$. Proto $\mathbf{k} * f(\mathbf{w}) \mapsto^* \mathbf{k} * \mathbf{u} \mapsto \mathbf{v}$, kde $\mathbf{v} \equiv (n+1) \cdot n! = (n+1)!$. $\square$

`Komentář:` Vidíte, jak „hutný“ a přitom formálně zcela přesný zápis důkazu naše formalizace umožňuje? Promyslete si podrobně všechny jeho kroky ještě jednou a dobře si uvědomte, co z čeho vyplývá a jak na sebe navazují.

Důkazy „neukončenosti“ výpočtů

Jinou otázkou je, jak zdůvodnit, že některý výpočet neskončí. K tomu využijeme následující tvrzení:

Věta 9.5. *Buď Δ deklarace. Pro každé $i \in \mathbb{N}$ definujeme relaci $\mapsto^i \subseteq \mathit{Exp}(\Delta) \times \mathit{Exp}(\Delta)$ předpisem $\mapsto^i\ =\ \underbrace{\mapsto \circ \cdots \circ \mapsto}_{i}$. Dále definitoricky klademe $\mapsto^0\ =\ \{(E,E) \mid E \in \mathit{Exp}(\Delta)\}$. Pak (relace „výpočet“) platí $\mapsto^*\ =\ \bigcup_{i=0}^{\infty} \mapsto^i$.*

Podle předchozí věty platí, že $E \mapsto^* F$ právě když $E \mapsto^i F$ pro nějaké $i \in \mathbb{N}$. Navíc musí existovat *nejmenší* i s touto vlastností. Toto pozorování bývá velmi užitečné v důkazech „neukončenosti“ výpočtů.

Příklad 9.6. *Uvažme deklaraci $f(x) = f(x)$.*

Věta. Pro každé $\mathbf{n} \in \mathit{Num}$ platí, že neexistuje žádné $\mathbf{m} \in \mathit{Num}$ takové, že $f(\mathbf{n}) \mapsto^* \mathbf{m}$.

`Důkaz` sporem: Předpokládejme, že existují $\mathbf{n}, \mathbf{m} \in \mathit{Num}$ takové, že $f(\mathbf{n}) \mapsto^* \mathbf{m}$. Pak existuje *nejmenší* $i \in \mathbb{N}$ takové, že $f(\mathbf{n}) \mapsto^i \mathbf{m}$. Jelikož výrazy $f(\mathbf{n})$ a $\mathbf{m}$ jsou různé, platí $i > 0$. Jelikož $\mapsto^i\ =\ \mapsto^{i-1} \circ \mapsto$ a $f(\mathbf{n}) \mapsto f(\mathbf{n})$, platí $f(\mathbf{n}) \mapsto^{i-1} \mathbf{m}$, což je spor s minimalitou i. $\square$

Rozšiřující studium

....

10 Důkazové postupy pro algoritmy

Úvod

Nyní si ukážeme, jak formální deklarativní jazyk z Lekce 9 využít k formálně přesným induktivním důkazům vybraných algoritmů. Dá se říci, že tato lekce je „vrcholem“ v naší snaze o matematické dokazování algoritmů v informatice.

Cíle

Na podkladě jednotlivých variant důkazů matematickou indukcí z Lekce 2 si ukážeme přehled formálních indukčních důkazových technik aplikovaných na vybrané algoritmy (v zápisech deklarativního jazyka).

10.1 Technika „fixace parametru“

Tato technika je vhodná pro případy, kdy je sice v algoritmu více parametrů, ale „zjevně“ dochází ke změně jen jednoho (nebo části) z nich a chování algoritmu ke zbylým „neměnným“ parametrům je dobře „předvídatelné“.

Příklad 10.1. *Uvažme deklaraci* Δ *obsahující pouze rovnici*

$$g(x,y) = \textbf{\textit{if}}\ x\ \textbf{\textit{then}}\ y + g(x-\mathbf{1},y)\ \textbf{\textit{else}}\ \mathbf{0}\,.$$

Věta. Pro každé $m,n \in \mathbb{N}$ platí $g(\mathbf{m},\mathbf{n}) \mapsto^* \mathbf{z}$, kde $\mathbf{z} \equiv m \cdot n$.

`Důkaz`: Budiž $n \in \mathbb{N}$ libovolné ale pro další úvahy *pevné*. Dokážeme, že pro každé $m \in \mathbb{N}$ platí $g(\mathbf{m},\mathbf{n}) \mapsto^* \mathbf{z}$, kde $\mathbf{z} \equiv m \cdot n$, indukcí vzhledem k m.

* *Báze* $m = 0$. Platí $g(\mathbf{0},\mathbf{n}) \mapsto \textbf{if 0 then n} + g(\mathbf{0}-\mathbf{1},\mathbf{n})\ \textbf{else 0} \mapsto \mathbf{0}$.
* *Indukční krok*. Nechť $m+1 \equiv \mathbf{k}$. Pak

  $$g(\mathbf{k},\mathbf{n}) \mapsto \textbf{if k then n} + g(\mathbf{k}-\mathbf{1},\mathbf{n})\ \textbf{else 0} \mapsto \mathbf{n} + g(\mathbf{k}-\mathbf{1},\mathbf{n}) \mapsto \mathbf{n} + g(\mathbf{w},\mathbf{n})\,,$$

  kde je $\mathbf{w} \equiv m$. Podle I.P. platí $g(\mathbf{w},\mathbf{n}) \mapsto^* \mathbf{u}$ pro $\mathbf{u} \equiv m \cdot n$. Dále $\mathbf{n} + g(\mathbf{w},\mathbf{n}) \mapsto^* \mathbf{n}+\mathbf{u} \mapsto \mathbf{v}$, kde $\mathbf{v} \equiv n + (m \cdot n) = (m+1)\cdot n = k \cdot n$, a tím jsme dohromady hotovi s důkazem $g(\mathbf{k},\mathbf{n}) \mapsto^* \mathbf{v}$. □

10.2 Technika „indukce k součtu parametrů“

Toto lze použít především v případech, kdy se v průběhu algoritmu vždy některý parametr zmenšuje, ale pokaždé je to některý jiný parametr, takže v indukci se nelze zaměřit jen na jeden z nich.

Příklad 10.2. *Uvažme deklaraci* Δ *obsahující pouze rovnici*

$$g(x,y) = \textbf{\textit{if}}\ x\ \textbf{\textit{then}}\ (\textbf{\textit{if}}\ y\ \textbf{\textit{then}}\ g(x-\mathbf{1},y) + g(x,y-\mathbf{1})\ \textbf{\textit{else}}\ \mathbf{0})\ \textbf{\textit{else}}\ \mathbf{0}\,.$$

Věta. Pro každé $m,n \in \mathbb{N}$ platí $g(\mathbf{m},\mathbf{n}) \mapsto^* \mathbf{0}$.

Tvrzení této věty přímo nelze dokázat indukcí vzhledem k m, ani indukcí vzhledem k n, neboť u žádného z m, n nemáme zaručeno, že se vždy zmenší. Důkaz lze ovšem postavit na faktu, že se vždy zmenší alespoň jeden z m, n, neboli se vždy zmenší součet m a n. To znamená, že výše uvedené tvrzení nejprve přeformulujeme do následující (matematicky ekvivalentní) podoby:

> **Věta**. Pro každé $i \in \mathbb{N}$ platí, že jestliže $i = m + n$ pro kterákoliv $m, n \in \mathbb{N}$, pak $g(\mathbf{m}, \mathbf{n}) \mapsto^* \mathbf{0}$.

`Důkaz` indukcí vzhledem k i: *Báze* $i = 0$ znamená, že $0 = m + n$ pro $m, n \in \mathbb{N}$, neboli $m = n = 0$. Dokazujeme tedy, že $g(\mathbf{0}, \mathbf{0}) \mapsto^* \mathbf{0}$. Platí

$$g(\mathbf{0}, \mathbf{0}) \mapsto \textbf{if } \mathbf{0} \textbf{ then } (\textbf{if } \mathbf{0} \textbf{ then } g(\mathbf{0} - \mathbf{1}, \mathbf{0}) + g(\mathbf{0}, \mathbf{0} - \mathbf{1}) \textbf{ else } \mathbf{0}) \textbf{ else } \mathbf{0} \mapsto \mathbf{0}\,.$$

Indukční krok. Nechť $i+1 = m+n$, kde $m, n \in \mathbb{N}$. Nyní rozlišíme tři možnosti (z nichž první dvě jsou svým způsobem jen rozšířeními předchozí báze indukce):

* Pro $m = 0$ platí

$$g(\mathbf{0}, \mathbf{n}) \mapsto \textbf{if } \mathbf{0} \textbf{ then } (\textbf{if } \mathbf{n} \textbf{ then } g(\mathbf{0} - \mathbf{1}, \mathbf{n}) + g(\mathbf{0}, \mathbf{n} - \mathbf{1}) \textbf{ else } \mathbf{0}) \textbf{ else } \mathbf{0} \mapsto \mathbf{0}\,.$$

* Pro $m > 0, n = 0$ platí

$$g(\mathbf{m}, \mathbf{0}) \mapsto \textbf{if } \mathbf{m} \textbf{ then } (\textbf{if } \mathbf{0} \textbf{ then } g(\mathbf{m} - \mathbf{1}, \mathbf{0}) + g(\mathbf{m}, \mathbf{0} - \mathbf{1}) \textbf{ else } \mathbf{0}) \textbf{ else } \mathbf{0} \mapsto$$
$$\mapsto \textbf{if } \mathbf{0} \textbf{ then } g(\mathbf{m} - \mathbf{1}, \mathbf{0}) + g(\mathbf{m}, \mathbf{0} - \mathbf{1}) \textbf{ else } \mathbf{0} \mapsto \mathbf{0}\,.$$

* Pro $m > 0, n > 0$ platí

$$g(\mathbf{m}, \mathbf{n}) \mapsto \textbf{if } \mathbf{m} \textbf{ then } (\textbf{if } \mathbf{n} \textbf{ then } g(\mathbf{m} - \mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n} - \mathbf{1}) \textbf{ else } \mathbf{0}) \textbf{ else } \mathbf{0} \mapsto$$
$$\mapsto \textbf{if } \mathbf{n} \textbf{ then } g(\mathbf{m} - \mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n} - \mathbf{1}) \textbf{ else } \mathbf{0} \mapsto g(\mathbf{m} - \mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n} - \mathbf{1})\,.$$

  Podle I.P. platí $g(\mathbf{m} - \mathbf{1}, \mathbf{n}) \mapsto^* \mathbf{0}$ a současně $g(\mathbf{m}, \mathbf{n} - \mathbf{1}) \mapsto^* \mathbf{0}$, proto

$$g(\mathbf{m} - \mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n} - \mathbf{1}) \mapsto^* \mathbf{0} + g(\mathbf{m}, \mathbf{n} - \mathbf{1}) \mapsto^* \mathbf{0} + \mathbf{0} \mapsto \mathbf{0}\,.$$

Tím jsme s důkazem matematickou indukcí hotovi. □

Zajímavější verze

Udělejme si předchozí nudný příklad trochu zajímavějším (ale co se týče důkazu stále v podstatě stejným. . .).

Příklad 10.3. *Uvažme deklaraci* Δ *obsahující pouze rovnici*

$$g(x, y) = \textbf{\textit{if} } x \textbf{ \textit{then} } (\textbf{\textit{if} } y \textbf{ \textit{then} } g(x - \mathbf{1}, y) + g(x, y - \mathbf{1}) \textbf{ \textit{else} } \mathbf{1}) \textbf{ \textit{else} } \mathbf{1}\,.$$

> **Věta**. Pro každé $m, n \in \mathbb{N}$ platí $g(\mathbf{m}, \mathbf{n}) \mapsto^* \mathbf{k}$, kde $k = \binom{m+n}{m}$ (kombinační číslo).

Toto tvrzení opět budeme dokazovat indukcí vzhledem k $i = m + n$.

Vzpomeňte si nejprve na známý *Pascalův trojúhelník* kombinačních čísel, který je definovaný rekurentním vztahem

$$\binom{a+1}{b+1} = \binom{a}{b+1} + \binom{a}{b}.$$

Nepřipomíná to trochu naši deklaraci? Je však třeba správně „nastavit“ význam parametrů a, b.

`Důkaz` indukcí vzhledem k i: *Báze* $i = 0$ znamená, že $0 = m + n$ pro $m, n \in \mathbb{N}$, neboli $m = n = 0$. Dokazujeme tedy, že $g(\mathbf{0}, \mathbf{0}) \mapsto^* \mathbf{1}$. Platí

$$g(\mathbf{0}, \mathbf{0}) \mapsto \mathbf{if\ 0\ then\ (if\ 0\ then\ } g(\mathbf{0}-\mathbf{1}, \mathbf{0}) + g(\mathbf{0}, \mathbf{0}-\mathbf{1}) \mathbf{\ else\ 1)\ else\ 1} \mapsto \mathbf{1}\,.$$

Indukční krok. Nechť $i+1 = m+n$, kde $m, n \in \mathbb{N}$. Opět rozlišíme stejné tři možnosti:

$*$ Pro $m = 0$ platí

$$g(\mathbf{0}, \mathbf{n}) \mapsto \mathbf{if\ 0\ then\ (if\ n\ then\ } g(\mathbf{0}-\mathbf{1}, \mathbf{n}) + g(\mathbf{0}, \mathbf{n}-\mathbf{1}) \mathbf{\ else\ 1)\ else\ 1} \mapsto \mathbf{1}\,.$$

$*$ Pro $m > 0, n = 0$ platí

$$g(\mathbf{m}, \mathbf{0}) \mapsto \mathbf{if\ m\ then\ (if\ 0\ then\ } g(\mathbf{m}-\mathbf{1}, \mathbf{0}) + g(\mathbf{m}, \mathbf{0}-\mathbf{1}) \mathbf{\ else\ 1)\ else\ 1} \mapsto$$

$$\mapsto \mathbf{if\ 0\ then\ } g(\mathbf{m}-\mathbf{1}, \mathbf{0}) + g(\mathbf{m}, \mathbf{0}-\mathbf{1}) \mathbf{\ else\ 1} \mapsto \mathbf{1}\,.$$

$*$ Pro $m > 0, n > 0$ platí

$$g(\mathbf{m}, \mathbf{n}) \mapsto \mathbf{if\ m\ then\ (if\ n\ then\ } g(\mathbf{m}-\mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n}-\mathbf{1}) \mathbf{\ else\ 1)\ else\ 1} \mapsto$$

$$\mapsto \mathbf{if\ n\ then\ } g(\mathbf{m}-\mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n}-\mathbf{1}) \mathbf{\ else\ 1} \mapsto g(\mathbf{m}-\mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n}-\mathbf{1})\,.$$

Podle I.P. platí $g(\mathbf{m}-\mathbf{1}, \mathbf{n}) \mapsto^* \mathbf{k_1}$, kde $\mathbf{k_1} \equiv \binom{m+n-1}{m-1}$, a současně $g(\mathbf{m}, \mathbf{n}-\mathbf{1}) \mapsto^* \mathbf{k_2}$, kde $\mathbf{k_2} \equiv \binom{m+n-1}{m}$. Přitom z Pascalova trojúhelníka plyne

$$\binom{m+n-1}{m-1} + \binom{m+n-1}{m} = \binom{m+n-1+1}{m} = \binom{m+n}{m},$$

a proto

$$g(\mathbf{m}-\mathbf{1}, \mathbf{n}) + g(\mathbf{m}, \mathbf{n}-\mathbf{1}) \mapsto^* \mathbf{k_1} + \mathbf{k_2} \mapsto^* \mathbf{k} \equiv \binom{m+n}{m}.$$

□

10.3 Technika „zesílení dokazovaného tvrzení“

Velmi častou situací při dokazování algoritmu je, že se zajímáme o hodnoty některých proměnných nebo „výstupy“ některé funkce, ale ke správnému matematickému důkazu musíme „postihnout“ i chování jiných funkcí a proměnných v algoritmu. Taková situace pak typicky vede na potřebu *zesílení požadovaného tvrzení* v matematické indukci.

Příklad 10.4. *Uvažme deklaraci* Δ *obsahující tyto rovnice:*

$f(x) = \textbf{\textit{if}}\ x\ \textbf{\textit{then}}\ h(x)\ \textbf{\textit{else}}\ \mathbf{1}$
$h(x) = \textbf{\textit{if}}\ x\ \textbf{\textit{then}}\ f(x-\mathbf{1}) + h(x-\mathbf{1})\ \textbf{\textit{else}}\ \mathbf{1}$

Věta. Pro každé $n \in \mathbb{N}$ platí $f(\mathbf{n}) \mapsto^* \mathbf{m}$, kde $m = 2^n$.

Požadované tvrzení bohužel nelze přímo dokázat indukcí podle n. Řešením je přeformulování dokazovaného tvrzení do silnější podoby, kterou již indukcí dokázat lze:

Věta. *Pro každé $n \in \mathbb{N}$ platí $f(\mathbf{n}) \mapsto^* \mathbf{m}$ a $h(\mathbf{n}) \mapsto^* \mathbf{m}$, kde $m = 2^n$.*

`Důkaz`, již poměrně snadno indukcí vzhledem k n:

* *Báze* $n = 0$. Platí

$$f(\mathbf{0}) \mapsto \textbf{if 0 then } h(\mathbf{0}) \textbf{ else 1} \mapsto \mathbf{1},$$
$$h(\mathbf{0}) \mapsto \textbf{if 0 then } f(\mathbf{0}-\mathbf{1}) + h(\mathbf{0}-\mathbf{1}) \textbf{ else 1} \mapsto \mathbf{1}.$$

* *Indukční krok:* Nechť $n + 1 \equiv \mathbf{k}$, pak platí

$$f(\mathbf{k}) \mapsto \textbf{if k then } h(\mathbf{k}) \textbf{ else 1} \mapsto h(\mathbf{k}) \mapsto$$
$$\mapsto \textbf{if k then } f(\mathbf{k}-\mathbf{1}) + h(\mathbf{k}-\mathbf{1}) \textbf{ else 1} \mapsto f(\mathbf{k}-\mathbf{1}) + h(\mathbf{k}-\mathbf{1}) \mapsto f(\mathbf{w}) + h(\mathbf{k}-\mathbf{1}),$$

kde $\mathbf{w} \equiv k - 1 = n$. Podle I.P. platí $f(\mathbf{w}) \mapsto^* \mathbf{m}$, kde $m = 2^n$. Zároveň také (naše „*zesílení*“) platí i $h(\mathbf{w}) \mapsto^* \mathbf{m}$, a proto

$$f(\mathbf{w}) + h(\mathbf{w}) \mapsto^* \mathbf{m} + h(\mathbf{w}) \mapsto^* \mathbf{m} + \mathbf{m} \mapsto \mathbf{q},$$

kde $q = m + m = 2m = 2 \cdot 2^n = 2^{n+1} = 2^k$. Proto tranzitivně $f(\mathbf{k}) \mapsto \mathbf{q}$ a první část našeho tvrzení platí i pro $n + 1 \equiv \mathbf{k}$.

Podobně je třeba *ještě dokončit* druhou část tvrzení.

$$h(\mathbf{k}) \mapsto \textbf{if k then } f(\mathbf{k}-\mathbf{1}) + h(\mathbf{k}-\mathbf{1}) \textbf{ else 1} \mapsto$$
$$f(\mathbf{k}-\mathbf{1}) + h(\mathbf{k}-\mathbf{1}) \mapsto^* f(\mathbf{w}) + h(\mathbf{k}-\mathbf{1}),$$

kde $\mathbf{w} \equiv k - 1 = n$. Podle I.P. platí $f(\mathbf{w}) \mapsto^* \mathbf{m}$, kde $m = 2^n$, a také $h(\mathbf{w}) \mapsto^* \mathbf{m}$, a proto

$$f(\mathbf{w}) + h(\mathbf{w}) \mapsto^* \mathbf{m} + \mathbf{m} \mapsto \mathbf{q},$$

kde $q = m + m = 2 \cdot 2^n = 2^{n+1} = 2^k$. Proto $h(\mathbf{k}) \mapsto \mathbf{q}$ a i druhá část našeho tvrzení platí pro $n + 1 \equiv \mathbf{k}$. □

10.4 Dva „klasické“ algoritmy

Již z dávných dob antiky pochází tento zajímavý a koneckonců velmi jednoduchý algoritmus teorie čísel. (Víte, jestli tehdy vůbec mluvili o algoritmech a programech nebo jako to bylo?)

Euklidův algoritmus

Věta 10.5. *Uvažme deklaraci Δ obsahující pouze rovnici*

$$g(x,y) = \textbf{\textit{if}}\ x-y\ \textbf{\textit{then}}\ g(x-y,y)\ \textbf{\textit{else}}\ (\textbf{\textit{if}}\ y-x\ \textbf{\textit{then}}\ g(x,y-x)\ \textbf{\textit{else}}\ x)\,.$$

Pak pro každé nenulové $m,n \in \mathbb{N}$ platí $g(\mathbf{m},\mathbf{n}) \mapsto^* \mathbf{z}$, kde z je největší společný dělitel čísel m,n.

`Důkaz` indukcí k $i = m+n$. (Tj. dokazujeme následující tvrzení: *Pro každé $i \geq 2$ platí, že jestliže $i = m+n$, kde $m,n \in \mathbb{N}$, $m,n > 0$, pak z je největší společný dělitel čísel m,n.*)

V *bázi* pro $i = 2$ je $m,n = 1$ a platí

$$\begin{aligned}
&g(\mathbf{1},\mathbf{1}) \mapsto \textbf{if } \mathbf{1}-\mathbf{1} \textbf{ then } g(\mathbf{1}-\mathbf{1},\mathbf{1}) \textbf{ else } (\textbf{if } \mathbf{1}-\mathbf{1} \textbf{ then } g(\mathbf{1},\mathbf{1}-\mathbf{1}) \textbf{ else } \mathbf{1}) \mapsto\\
&\mapsto \textbf{if } \mathbf{0} \textbf{ then } g(\mathbf{1}-\mathbf{1},\mathbf{1}) \textbf{ else } (\textbf{if } \mathbf{1}-\mathbf{1} \textbf{ then } g(\mathbf{1},\mathbf{1}-\mathbf{1}) \textbf{ else } \mathbf{1}) \mapsto\\
&\mapsto \textbf{if } \mathbf{1}-\mathbf{1} \textbf{ then } g(\mathbf{1},\mathbf{1}-\mathbf{1}) \textbf{ else } \mathbf{1} \mapsto \textbf{if } \mathbf{0} \textbf{ then } g(\mathbf{1},\mathbf{1}-\mathbf{1}) \textbf{ else } \mathbf{1} \mapsto \mathbf{1}\,.
\end{aligned}$$

Indukční krok. Nechť $i+1 = m+n$ kde $m,n \in \mathbb{N}$. Probereme tři možnosti:

* $m = n$. Pak

$$\begin{aligned}
&g(\mathbf{m},\mathbf{n}) \mapsto \textbf{if } \mathbf{m}-\mathbf{n} \textbf{ then } g(\mathbf{m}-\mathbf{n},\mathbf{n}) \textbf{ else } (\textbf{if } \mathbf{n}-\mathbf{m} \textbf{ then } g(\mathbf{m},\mathbf{n}-\mathbf{m}) \textbf{ else } \mathbf{m}) \mapsto\\
&\textbf{if } \mathbf{0} \textbf{ then } g(\mathbf{m}-\mathbf{n},\mathbf{n}) \textbf{ else } (\textbf{if } \mathbf{n}-\mathbf{m} \textbf{ then } g(\mathbf{m},\mathbf{n}-\mathbf{m}) \textbf{ else } \mathbf{m}) \mapsto\\
&\textbf{if } \mathbf{n}-\mathbf{m} \textbf{ then } g(\mathbf{m},\mathbf{n}-\mathbf{m}) \textbf{ else } \mathbf{m} \mapsto \textbf{if } \mathbf{0} \textbf{ then } g(\mathbf{m},\mathbf{n}-\mathbf{m}) \textbf{ else } \mathbf{m} \mapsto \mathbf{m}\,.
\end{aligned}$$

* $m < n$. Pak

$$\begin{aligned}
&g(\mathbf{m},\mathbf{n}) \mapsto \textbf{if } \mathbf{m}-\mathbf{n} \textbf{ then } g(\mathbf{m}-\mathbf{n},\mathbf{n}) \textbf{ else } (\textbf{if } \mathbf{n}-\mathbf{m} \textbf{ then } g(\mathbf{m},\mathbf{n}-\mathbf{m}) \textbf{ else } \mathbf{m}) \mapsto\\
&\textbf{if } \mathbf{0} \textbf{ then } g(\mathbf{m}-\mathbf{n},\mathbf{n}) \textbf{ else } (\textbf{if } \mathbf{n}-\mathbf{m} \textbf{ then } g(\mathbf{m},\mathbf{n}-\mathbf{m}) \textbf{ else } \mathbf{m}) \mapsto\\
&\textbf{if } \mathbf{n}-\mathbf{m} \textbf{ then } g(\mathbf{m},\mathbf{n}-\mathbf{m}) \textbf{ else } \mathbf{m} \mapsto \textbf{if } \mathbf{z} \textbf{ then } g(\mathbf{m},\mathbf{n}-\mathbf{m}) \textbf{ else } \mathbf{m} \mapsto\\
&g(\mathbf{m},\mathbf{n}-\mathbf{m}) \mapsto g(\mathbf{m},\mathbf{k})\,,
\end{aligned}$$

  kde $\mathbf{k} \equiv n-m$. Platí $m+k = m+(n-m) = n \leq i$, takže podle I.P. také platí $g(\mathbf{m},\mathbf{k}) \mapsto^* \mathbf{z}$, kde z je největší společný dělitel čísel m a $n-m$. Ověříme, že z je největší společný dělitel čísel m a n.

  – Jelikož číslo z dělí čísla m a $n-m$, dělí i jejich součet $(n-m)+m = n$. Celkem z je společným dělitelem m a n.

  – Buď d nějaký společný dělitel čísel m a n. Pak d dělí také rozdíl $n-m$. Tedy d je společný dělitel čísel m a $n-m$. Jelikož z je největší společný dělitel čísel m a $n-m$, nutně d dělí z a závěr platí.

* $m > n$. Pak

$$g(\mathbf{m},\mathbf{n}) \mapsto^* g(\mathbf{m}-\mathbf{n},\mathbf{n}) \mapsto g(\mathbf{k},\mathbf{n})\,,$$

  kde $\mathbf{k} \equiv m-n$. Podle I.P. platí $g(\mathbf{k},\mathbf{n}) \mapsto^* \mathbf{z}$, kde z je největší společný dělitel čísel $m-n$ a n. Podobně jako výše ověříme, že z je také největší společný dělitel čísel m a n. □

`Poznámka`: Jak byste výše uvedený zápis Euklidova algoritmu vylepšili, aby správně „počítal" největšího společného dělitele i v případech, že $m = 0$ nebo $n = 0$?
Co v takových případech selže při současném zápise?

Inkrementace dekadického zápisu

Následuje příklad algoritmu, který všichni dobře znají už od dob základní školy – sčítání vícemístných čísel. My si ukážeme jeho zjednodušenou verzi coby *inkremenaci* čísla (tj. přičtení 1 k číslu).

> **Příklad 10.6.** *Mějme přirozené číslo m dekadicky zapsané pomocí číslic $(c_{k-1}c_{k-2}\dots c_1c_0)_{10}$ (kde zleva se implicitně vyplňují nuly). Pak dekadický zápis čísla $m' = m+1$ získáme takto:*

Algoritmus . *Inkrementace.*

```
k ← počet číslic m;
p ← 1;
for i ← 0,1,...,k−1,k do
    c'_i ← (c_i+p) mod 10;
    if c'_i ≠ 0 then p ← 0;
done
```

Zapišme tento kód formální deklarací našeho jazyka.

Řešení:

* Jelikož nyní nejsou k dispozici proměnné typu pole, „pomůžeme si“ funkčním zápisem číslic $g(i)$ a $g'(i)$ místo c_i, c'_i.
* Cyklus `for` nahradíme rekurzí (běžný postup).
* Nakonec „trikově“ nahradíme proměnnou p, která vyjadřuje *přenos* do i-tého řádu, zavedením nové funkce $p(i)$, což výrazně zjednoduší zápis deklarace.

Celá *formální deklarace* Δ bude vypadat například následovně:

$$g'(i) = (g(i) + p(i)) \bmod \mathbf{10}$$
$$p(i) \;=\; \textbf{if } i \textbf{ then } (\textbf{if } g'(i-1) \textbf{ then } \mathbf{0} \textbf{ else } \mathbf{1}) \textbf{ else } \mathbf{1}$$
$$g(\mathbf{0}) = \mathbf{c_0},\ g(\mathbf{1}) = \mathbf{c_1},\ \dots\ g(\mathbf{k-1}) = \mathbf{c_{k-1}}$$

Všimněte si zvláštního posledního řádku, kde jsou rovnice deklarující konstantní hodnoty jednotlivých číslic vstupního čísla m. (Proč to tak je zapsáno?)

> **Věta**. Pro každé $i \in \mathbb{N}$ platí, že $g'(i)$ udává dekadickou číslici i-tého řádu zprava čísla $m+1$, kde m má dekadický zápis po číslicích $(c_{k-1}\dots c_1c_0)_{10}$.

Dokažte si tvrzení sami za domácí úkol (diskutujte na IS). Je potřeba použít matematickou indukci se zesíleným předpokladem, který se bude vhodně vyjadřovat i o významu hodnoty $p(i)$ („*přenos*“). Pochopitelně je třeba pro úplnou správnost řešení ještě rozepsat operaci „*modulo*“ pomocí povolených aritmetických operací, což si také za úkol vyzkoušejte. □

Rozšiřující studium

....

11 Nekonečné množiny a zastavení algoritmu

Úvod

Bystrého čtenáře může snadno napadnout myšlenka, proč se vlastně zabýváme dokazováním zprávnosti algoritmů a programů, když by to přece (snad?) mohl za nás dělat automaticky počítač samotný. Bohužel to však nejde a je hlavním cílem této lekce ukázat důvody proč.

Konkrétně si dokážeme, že nelze algoritmicky rozhodnout, zda se daný algoritmus na svém vstupu zastaví nebo ne. Hlavními nástroji, které použijeme, budou nekonečné množiny a důkazová technika tzv. Cantorovy diagonály, která se ve velké míře používá právě v teoretické informatice. (Pro zvídavé; obdobně, ale mnohem složitěji, lze dokázat že ani matematické důkazy nelze obecně algoritmicky konstruovat...)

Cíle

Zavedeme si v „naivním pohledu“ nekonečné množiny a techniku důkazu Cantorovou diagonálou. Pak tuto techniku využijeme k důkazu algoritmické neřešitelnosti problému zastavení.

11.1 O kardinalitě a nekonečných množinách

`Definice`: Množina A je „nejvýše tak velká“ jako množina B, právě když existuje injektivní funkce $f : A \to B$. Množiny A a B jsou „stejně velké“ právě když mezi nimi existuje bijekce.

V případech nekonečných množin místo “velikosti” mluvíme formálně o jejich *kardinalitě*.

`Komentář:` Tyto definice kardinality množin „fungují“ dobře i pro nekonečné množiny.

* Například $\mathbb{N}$ a $\mathbb{Z}$ mají stejnou kardinalitu („stejně velké“), tzv. *spočetně nekonečné*).
* Lze snadno ukázat, že i $\mathbb{Q}$ je spočetně nekonečná, tj. existuje bijekce $f : \mathbb{N} \to \mathbb{Q}$.
* Existují ale i nekonečné množiny, které jsou „striktně větší“ než libovolná spočetná množina (příkladem je $\mathbb{R}$).
* Později dokážeme, že *existuje nekonečná posloupnost nekonečných množin, z nichž každá je striktně větší než všechny předchozí.*

Věta 11.1. *Neexistuje žádné surjektivní (tudíž ani bijektivní) zobrazení $g : \mathbb{N} \to \mathbb{R}$.*

Neformálně řečeno, reálných čísel je striktně více než přirozených.

`Důkaz` sporem. Nechť takové g existuje a pro zjednodušení se omezme jen na funkční hodnoty v intervalu $(0, 1)$. Podle hodnot zobrazení g si takto můžeme „uspořádat“ dekadické zápisy všech reálných čísel v intervalu $(0, 1)$ po řádcích do tabulky:

$$
\begin{array}{rcl}
g(0) = & 0. & 1\ 5\ 4\ 2\ 7\ 5\ 7\ 8\ 3\ 2\ 5\ \ldots \\
g(1) = & 0. & \ 2\ \ \ \ \ \ \ \ \ \ \ldots \\
g(2) = & 0. & \ \ 1\ \ \ \ \ \ \ \ \ \ldots \\
g(3) = & 0. & \ \ \ 3\ \ \ \ \ \ \ \ \ldots \\
g(4) = & 0. & \ \ \ \ 9\ \ \ \ \ \ \ \ldots \\
\vdots & \vdots & \ \ \ \ \ \ddots
\end{array}
$$

Nyní sestrojíme číslo $\alpha \in (0, 1)$ následovně; jeho i-tá číslice za desetinnou čárkou bude 1, pokud v i-tém řádku tabulky na diagonále není 1, jinak to bude 2. V našem příkladě $\alpha = 0.21211\ldots$

Kde se naše číslo α v tabulce nachází? (Nezapomeňme, g byla surjektivní, takže tam α musí být!) Kostrukce však ukazuje, že α se od každého čísla v tabulce liší na aspoň jednom desetinném místě, to je spor. *(Až na drobný technický detail s rozvojem* $\ldots\bar{9}$.*)* $\square$

V obecnosti lze dokonce analogickým způsobem dokázat následovné.

Věta 11.2. *Buď M libovolná množina. Pak existuje injektivní zobrazení $f : M \to 2^M$, ale neexistuje žádné bijektivní zobrazení $g : M \to 2^M$.*

Důkaz: Dokážeme nejprve existenci f. Stačí ale položit $f(x) = \{x\}$ pro každé $x \in M$. Pak $f : M \to 2^M$ je zjevně injektivní.

Neexistenci g dokážeme sporem. Předpokládejme tedy naopak, že existuje bijekce $g : M \to 2^M$. Uvažme množinu $K \subseteq M$ definovanou takto:

$$K = \{x \in M \mid x \notin g(x)\}.$$

Jelikož g je bijektivní a $K \in 2^M$, musí existovat $x \in M$ takové, že $g(x) = K$. Nyní rozlišíme dvě možnosti:

– $x \in g(x)$. Tj. $x \in K$. Pak ale $x \notin g(x)$ z definice K, spor.

– $x \notin g(x)$. To podle definice K znamená, že $x \in K$, tj. $x \in g(x)$, spor. $\square$

Komentář: Dvě navazující poznámky.

- Z toho, že nekonečna mohou být „různě velká“, lze lehce odvodit řadu dalších faktů. V jistém smyslu je např. množina všech „problémů“ větší než množina všech algoritmů (obě množiny jsou nekonečné), proto nutně existují problémy, které *nejsou algoritmicky řešitelné*.
- Technika použitá v důkazech *Vět 11.1 a 11.2* se nazývá *Cantorova diagonální metoda*, nebo také zkráceně *diagonalizace*.

  Konstrukci množiny K lze znázornit pomocí následující tabulky:

	a	b	c	d	$\cdots$
$g(a)$	$\surd$	$-$	$-$	$\surd$	$\cdots$
$g(b)$	$\surd$	$-$	$-$	$\surd$	$\cdots$
$g(c)$	$-$	$\surd$	$-$	$\surd$	$\cdots$
$g(d)$	$-$	$-$	$\surd$	$\surd$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

  Symbol $\surd$ resp. $-$ říká, že prvek uvedený v záhlaví sloupce patří resp. nepatří do množiny uvedené v záhlaví řádku. Tedy např. $d \in g(b)$ a $a \notin g(d)$.

„Naivní“ množinové paradoxy

Naivní teorie množin, jak jsme si ji uvedli i v tomto předmětu, trpí mnoha neduhy a nepřesnostmi, které vyplynou na povrch především při „neopatrné manipulaci“ s (nekonečnými) množinami. Abychom se těmto „neopatrnostem“ vyhnuli bez přílišné formalizace, dva základní z těchto paradoxů si nyní ukážeme.

Příklad 11.3. *Uvážíme-li nyní nekonečnou posloupnost množin*

$$A_1, A_2, A_3, \cdots$$

kde $A_1 = \mathbb{N}$ *a* $A_{i+1} = 2^{A_i}$ *pro každé* $i \in \mathbb{N}$, *je vidět, že všechny množiny jsou nekonečné a každá je „striktně větší" než libovolná předchozí.*

Kde však v tomto řazení mohutností bude „*množina všech množin*"?

Takto se koncem 19. století objevil první *Cantorův paradox* nově vznikající teorie množin. (Dnešní vysvětlení je jednoduché, prostě „množinu všech množin" nelze definovat, prostě v matematice neexistuje.) □

Brzy se však ukázalo, že je ještě *mnohem hůř*...

Russelův paradox

`Fakt`: Není pravda, že *každý soubor prvků* lze považovat za množinu.

* $X = \{M \mid M \text{ je množina taková, že } M \notin M\}$. Platí $X \in X$?
  - Ano. Tj. $X \in X$. Pak ale X splňuje $X \notin X$.
  - Ne. Pak X splňuje vlastnost $X \notin X$, tedy X je prvkem X, tj., $X \in X$.
* Obě možné odpovědi vedou ke sporu. X tedy nelze prohlásit za množinu.

`Komentář:` Vidíte zde podobnost přístupu s Cantorovou diagonalizací? Russelův paradox se objevil začátkem 20. století a jeho „jednoduchost" zasahující úplné základy matematiky (teorie množin) si vynutila hledání seriózního rozřešení a rozvoj formální matematické logiky.

Pro ilustraci tohoto paradoxu, slyšeli jste už, že „*v malém městečku žije holič, který holí právě ty muže městečka, kteří se sami neholí*"?

Zmíněné *paradoxy naivní teorie množin* zatím v tomto kurzu nerozřešíme, ale zapamatujeme si, že většina matematických a informatických disciplín vystačí s „*intuitivně bezpečnými*" *množinami.*

11.2 Algoritmická neřešitelnost problému zastavení

`Fakt`: Uvědomme si (velmi neformálně) několik základních myšlenek.

* Program v každém programovacím jazyce je konečná posloupnost složená z konečně mnoha symbolů (písmena, číslice, mezery, speciální znaky, apod.) Nechť Σ je množina všech těchto symbolů. Množina všech programů je tedy jistě podmnožinou množiny $\bigcup_{i\in\mathbb{N}} \Sigma^i$, která je spočetně nekonečná. Existuje tedy bijekce f mezi množinou $\mathbb{N}$ a množinou všech programů. Pro každé $i \in \mathbb{N}$ označme symbolem P_i program $f(i)$. Pro každý program P tedy existuje $j \in \mathbb{N}$ takové, že $P = P_j$.
* Každý možný vstup každého možného programu lze zapsat jako konečnou posloupnost symbolů z konečné mnočiny Γ. Množina všech možných vstupů je tedy spočetně nekonečná a existuje bijekce g mezi množinou $\mathbb{N}$ a množinou všech vstupů. Pro každé $i \in \mathbb{N}$ označme symbolem V_i vstup $g(i)$.
* Předpokládejme, že existuje program *Halt*, který pro dané $i, j \in \mathbb{N}$ zastaví s výstupem *ano*/*ne* podle toho, zda P_i pro vstup V_j zastaví, nebo ne.
* Tento předpoklad dále dovedeme ke sporu dokazujícímu, že problém zastavení nemá algoritmické řešení.

Tvrzení 11.4. *Předpoklad existence programu Halt vede ke sporu.*

Důkaz: Uvažme program *Diag* s následujícím kódem:

```
input k;
if  Halt(k,k) = ano  then  while true do ; done
```

(Program $Diag(k)$ má na rozdíl od *Halt* jen jeden vstup k, což bude důležité.)

Fungování programu *Diag* lze znázornit za pomocí následující tabulky:

	P_1	P_2	P_3	P_4	$\cdots$
V_1	$\surd$	$-$	$-$	$\surd$	$\cdots$
V_2	$\surd$	$-$	$-$	$\surd$	$\cdots$
V_3	$-$	$\surd$	$-$	$\surd$	$\cdots$
V_4	$-$	$-$	$\surd$	$\surd$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Symbol $\surd$ resp. $-$ říká, že program uvedený v záhlaví sloupce zastaví resp. nezastaví pro vstup uvedený v záhlaví řádku. Program *Diag* „zneguje“ diagonálu této tabulky.

Podle našeho předpokladu (*Diag* je program a posloupnost P_i zahrnuje všechny programy) existuje $j \in \mathbb{N}$ takové, že $Diag = P_j$. Zastaví *Diag* pro vstup V_j?

– *Ano*. Podle kódu *Diag* pak ale tento program vstoupí do nekonečné smyčky, tedy nezastaví.

– *Ne*. Podle kódu *Diag* pak ale `if` test neuspěje, a tento program tedy zastaví.

Předpoklad existence programu *Halt* tedy vede ke sporu. □

Komentář: Otázkami algoritmické (ne)řešitelnosti problémů se zabývá *teorie vyčíslitelnosti*. Metoda diagonalizace se také často využívá v *teorii složitosti* k důkazu toho, že dané dvě složitostní třídy jsou různé.

Rozšiřující studium

....

12 Délka výpočtu algoritmu

Úvod

Mimo samotné správnosti výsledku vypočteného zapsaným algoritmem je ještě jedno neméně důležité hledisko k posouzení vhodnosti algoritmu k řešení zadané úlohy. Jedná se o čas, který algoritmus stráví výpočtem.

Asi netřeba argumentovat, že přehnaně dlouhá doba odezvy programu je každému uživateli nepříjemná. A co třeba v real-time systémech, kde si zdržení prostě nemůžeme dovolit. Obligátní odpověď „kupme si rychlejší počítač“ bohužel není vždy řešením, jak při pokročilém studiu složitosti algoritmů sami poznáte. Mnohem větší potenciál zrychlení se skrývá v algoritmech samotných a jejich efektivním návrhu.

Cíle

V této lekci definujeme délku výpočtu algoritmu, přičemž definici postavíme na deklarativním jazyce z Lekce 9. Poté si ukážeme, jak se matematicky popisuje asymptotické chování funkcí, což se využívá především pro zjednodušené studium složitosti výpočtu algoritmu.

12.1 O významu délky výpočtu algoritmu

Uvažme deklarativní jazyk Definice 9.1.

`Definice`: Délkou výpočtu výrazu F nad deklarací Δ rozumíme nejmenší přirozené k takové, že pro něj existuje $\mathbf{m} \in \mathit{Num}$ pro něž $F \mapsto^k \mathbf{m}$. (Když takové $\mathbf{m}$ neexistuje, klademe $k = \infty$.)

`Komentář:` Jaká je délka výpočtu následujících výrazů?

* $\mathbf{3+4-5*6}$; Tři kroky $\mathbf{3+4-5*6} \mapsto \mathbf{3+4-30} \mapsto \mathbf{3+0} \mapsto \mathbf{3}$.
* $\mathbf{3+(5-4)*(6\div 2)}$; Tentokrát čtyři kroky $\mathbf{3+(5-4)*(6\div 2)} \mapsto \mathbf{3+1*(6\div 2)} \mapsto \mathbf{3+1*3} \mapsto \mathbf{3+3} \mapsto \mathbf{6}$.
* $\mathbf{2007}$; Žádný krok, tj. $k = 0$.

Příklad 12.1. *Pro ukázku uvažme deklaraci Δ obsahující pouze rovnici*

$$f(x) = \textbf{\textit{if}}\ x\ \textbf{\textit{then}}\ x * f(x-\mathbf{1})\ \textbf{\textit{else}}\ \mathbf{1}\,.$$

Věta. Pro každé $n \in \mathbb{N}$ je délka výpočtu výrazu $f(\mathbf{n})$ rovna $4n+2$.

`Důkaz` povedeme indukcí podle n:

- *Báze* $n = 0$. Platí $f(\mathbf{0}) \mapsto \textbf{if 0 then } \mathbf{0}*f(\mathbf{0}-\mathbf{1})\textbf{ else 1} \mapsto \mathbf{1}$, což jsou přesně 2 kroky, tj. $4 \cdot 0 + 2$.
- *Indukční krok*. Nechť $n+1 \equiv \mathbf{k}$. Pak

$$f(\mathbf{k}) \mapsto \textbf{if k then k} * f(\mathbf{k}-\mathbf{1})\textbf{ else 1} \mapsto \mathbf{k} * f(\mathbf{k}-\mathbf{1}) \mapsto \mathbf{k} * f(\mathbf{w})\,,$$

kde $\mathbf{w} \equiv k - 1 = n$. To jsou přesně 3 kroky. Podle I.P. je délka výpočtu výrazu $f(\mathbf{w})$ rovna $4n+2$. Poté následuje ještě jeden poslední krok vynásobení $\mathbf{k}$. Celkem se provedlo $3 + 4n + 2 + 1 = 4(n+1) + 2 = 4k + 2$ kroků. □

Počítat přesně nebo raději ne?

`Komentář:` Jaký má smysl určení přesného počtu kroků algoritmu při dnešních CPU? Copak jsme dnes schopni jednoznačně říci, jak dlouho jedna instrukce CPU trvá? Z druh strany, i když víme, že algoritmus A třeba potřebuje $2n$ kroků výpočtu a algoritmus B třeba potřebuje $3n$ kroků, je mezi nimi až takový rozdíl? Stačí, když B spustíme na dvakrát rychlejším počítači a poměr se hned obrátí. Obě tyto prakticky motivované úvahy nás povedou k poznání, že aditivní a multiplikativní faktory funkce počtu kroků algoritmu jsou vlastně zanedbatelné.

12.2 Asymptotické značení a odhady funkcí

Zajímá-li nás jen rychlost růstu funkce $f(n)$ v závislosti na n, zaměřujeme se především na tzv. *asymptotické chování* f při velkých hodnotách n. V popisu f nás tedy nezajímají ani různé přičtené "drobné členy", které se významněji projevují jen pro malá n, ani konstanty, kterými je f násobena a které jen ovlivňují číselnou hodnotu $f(n)$, ale ne rychlost růstu.

`Komentář`: Tak například funkce $f(n) = n^2$ roste (zhruba) stejně rychle jako $f'(n) = 100000000n^2$ i jako $f''(n) = 0.00000001n^2 - 100000000n - 1000000$. Naopak $h(n) = 0.00000000001n^3$ roste mnohem rychleji než $f'(n) = 100000000n^2$.

Pro porovnávání rychlostí růstů funkcí nám slouží následující terminologie.

`Definice`: Nechť $g : \mathbb{N} \to \mathbb{N}$ je daná funkce. Pro funkci $f : \mathbb{N} \to \mathbb{N}$ píšeme

$$f \in O(g)$$

pokud existují konstanty $A, B > 0$ takové, že

$$\forall n \in \mathbb{N} : f(n) \leq A \cdot g(n) + B\,.$$

V praxi se obvykle (i když matematicky méně přesně) píše místo $f \in O(g)$ výraz

$$f(n) = O(g(n))\,.$$

Znamená to, slovně řečeno, že funkce f *neroste rychleji* než funkce g. (I když pro malá n třeba může být $f(n)$ mnohem větší než $g(n)$.)

`Poznámka`: Kromě vlastnosti $f \in O(g)$ se někdy setkáte i s vlastností $f \in o(g)$, která znamená $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (funkce f *roste striktně pomaleji* než g).

`Definice`: Píšeme $f \in \Omega(g)$, neboli $f(n) = \Omega(g(n))$, pokud $g \in O(f)$. Dále píšeme $f \in \Theta(g)$, neboli $f(n) = \Theta(g(n))$, pokud $f \in O(g)$ a zároveň $f \in \Omega(g)$, neboli $g \in O(f)$.

Výraz $f(n) = \Theta(g(n))$ pak čteme jako "funkce f *roste stejně rychle* jako funkce g".

`Značení`: O funkci $f(n)$ říkáme:

* $f(n) = \Theta(n)$ je *lineární* funkce,
* $f(n) = \Theta(n^2)$ je *kvadratická* funkce,
* $f(n) = \Theta(\log n)$ je *logaritmická* funkce,
* $f(n) = O(n^c)$ pro nějaké $c > 0$ je *polynomiální* funkce,
* $f(n) = \Theta(c^n)$ pro nějaké $c > 1$ je *exponenciální* funkce.

Příklad 12.2. *(opakovaný) Zjistěte, kolik znaků* `'x'` *v závislosti na celočíselné hodnotě n vstupního parametru* `n` *vypíše následující algoritmus.*

Algoritmus 12.3.

```
for  i ← 1,2,3,...,n-1,n  do
    for  j ← 1,2,3,...,i-1,i  do
        vytiskni  'x';
    done
done
```

Zatímco v Lekci 8 jsme trochu zdlouhavě indukcí dokazovali, že výsledkem je $\frac{1}{2}n(n+1)$ `'x'`, nyní si mnohem snadněji odvodíme, že počet `'x'` je $\Theta(n^2)$, což je dostačující asymptotická odpověď ve většině informatických aplikací.

`Důkaz`: Shora hned odhadneme, že každá z n iterací vnějšího cyklu vytiskne $i \leq n$ znaků `'x'`, takže celkem je nejvýše n^2 `'x'`. Naopak zdola hned vidíme, že posledních $n/2$ iterací vnějšího cyklu vytiskne $i \geq n/2$ znaků `'x'`, takže celkem je alespoň $(n/2)\cdot(n/2) = n^2/4$ `'x'`. Z toho podle definice hned vyjde asymptotický odhad $\Theta(n^2)$. □

Příklad 12.4. *Příklady růstů různých funkcí.*

Funkce $\boldsymbol{f(n) = \Theta(n)}$: pokud n vzroste na dvojnásobek, tak hodnota $f(n)$ taktéž vzroste (zhruba) na dvojnásobek. To platí jak pro funkci $f(n) = n$, tak i pro $1000000000n$ nebo $n + \sqrt{n}$, atd.

Funkce $\boldsymbol{f(n) = \Theta(n^2)}$: pokud n vzroste na dvojnásobek, tak hodnota $f(n)$ vzroste (zhruba) na čtyřnásobek. To platí jak pro funkci $f(n) = n^2$, tak i pro $1000n^2 + 1000n$ nebo $n^2 - 99999999n - 99999999$, atd.

Naopak pro funkci $\boldsymbol{f(n) = \Theta(2^n)}$: pokud n vzroste byť jen o 1, tak hodnota $f(n)$ už vzroste (zhruba) na dvojnásobek. To je obrovský rozdíl exponenciálních proti polynomiálním funkcím.

Pokud vám třeba funkce $\boldsymbol{999999n^2}$ připadá velká, jak stojí ve srovnání s 2^n? Zvolme třeba $n = 1000$, kdy $999999n^2 = 999999000000$ je ještě rozumně zapsatelné číslo, ale $2^{1000} \simeq 10^{300}$ byste už na řádek nenapsali. Pro $n = 10000$ je rozdíl ještě mnohem výraznější! □

Rekurentní odhady

V tomto oddíle si uvedeme krátký přehled některých rekurentních vzorců, se kterými se můžete setkat při řešení časové složitosti (převážně rekurzivních) algoritmů.

Lema 12.5. *Nechť $a_1, \ldots, a_k, c > 0$ jsou kladné konstanty takové, že $a_1 + \ldots + a_k < 1$, a funkce $T : \mathbb{N} \to \mathbb{N}$ splňuje nerovnost*

$$T(n) \leq T(\lceil a_1 n \rceil) + T(\lceil a_2 n \rceil) + \ldots + T(\lceil a_k n \rceil) + cn\,.$$

Pak $T(n) = O(n)$.

`Důkaz`: Zvolme $\varepsilon > 0$ takové, že $a_1 + \ldots + a_k < 1 - 2\varepsilon$. Pak pro dostatečně velká n platí (i se zaokrouhlením nahoru) $\lceil a_1 n \rceil + \ldots + \lceil a_k n \rceil \leq (1-\varepsilon)n$, řekněme pro všechna $n \geq n_0$. Dále zvolme dostatečně velké $d > 0$ tak, že $\varepsilon d > c$ a zároveň $d > \max\{\frac{1}{n}T(n) : n = 1, \ldots, n_0\}$.

Dále už snadno indukcí podle n dokážeme $T(n) \leq dn$ pro všechna $n \geq 1$:

- Pro $n \leq n_0$ je $T(n) \leq dn$ podle naší volby d.
- Předpokládejme, že $T(n) \leq dn$ platí pro všechna $n < n_1$, kde $n_1 > n_0$ je libovolné. Nyní dokážeme i pro n_1

$$T(n_1) \leq T(\lceil a_1 n_1 \rceil) + \ldots + T(\lceil a_k n_1 \rceil) + cn_1 \leq$$
$$\leq d \cdot \lceil a_1 n_1 \rceil + \ldots + d \cdot \lceil a_k n_1 \rceil + cn_1 \leq$$
$$\leq d \cdot (1-\varepsilon)n_1 + cn_1 \leq dn_1 - (\varepsilon d - c)n_1 \leq dn_1\,.$$

□

Lema 12.6. *Nechť $k \geq 2$ a $a_1, \ldots, a_k, c > 0$ jsou kladné konstanty takové, že $a_1 + \ldots + a_k = 1$, a funkce $T : \mathbb{N} \to \mathbb{N}$ splňuje nerovnost*

$$T(n) \leq T(\lceil a_1 n \rceil) + T(\lceil a_2 n \rceil) + \ldots + T(\lceil a_k n \rceil) + cn \,. \tag{1}$$

Pak $T(n) = O(n \cdot \log n)$.

`Důkaz` (nematematický náznak): Bylo by možno postupovat obdobně jako v předchozím důkaze, ale výpočty by byly složitější. Místo formálního důkazu indukcí nyní předestřeme poměrně jednoduchou úvahu zdůvodňující řešení $T(n) = O(n \cdot \log n)$.

Představme si, že upravujeme pravou stranu výrazu (1) v následujících krocích: V každém kroku rozepíšeme každý člen $T(m)$ s dostatečně velkým argumentem m rekurzivní aplikací výrazu (1) (s $T(m)$ na levé straně). Jelikož $a_1 + \ldots + a_k = 1$, součet hodnot argumentů všech $T(\cdot)$ ve zpracovávaném výrazu bude stále zhruba n. Navíc po zhruba $t = \Theta(\log n)$ krocích už budou hodnoty argumentů všech $T(\cdot)$ "malé" (nebude dále co rozepisovat), neboť $0 < a_i < 1$ a tudíž $a_i^t \cdot n < 1$ pro všechna i. Při každém z kroků našeho rozpisu se ve výrazu (1) přičte hodnota $cn = O(n)$, takže po t krocích bude výsledná hodnota

$$T(n) = t \cdot O(n) + O(n) = O(n \cdot \log n) \,.$$

Vyzkoušejte si tento postup sami na konkrétním příkladě $T'(n) \leq 2T'\left(\frac{n}{2}\right) + n$. □

V obecnosti je známo:

Lema 12.7. *Nechť $a \geq 1$, $b > 1$ jsou konstanty, $f : \mathbb{N} \to \mathbb{N}$ je funkce a pro funkci $T : \mathbb{N} \to \mathbb{N}$ platí rekurentní vztah*

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + f(n) \,.$$

Pak platí:

- ∗ *Je-li $f(n) = O(n^c)$ a $c < \log_b a$, pak $T(n) = O(n^{\log_b a})$.*
- ∗ *Je-li $f(n) = \Theta(n^{\log_b a})$, pak $T(n) = O(n^{\log_b a} \cdot \log n)$.*
- ∗ *Je-li $f(n) = \Theta(n^c)$ a $c > \log_b a$, pak $T(n) = O(n^c)$.*

Důkaz tohoto obecného tvrzení přesahuje rozsah našeho předmětu. Všimněte si, že nikde ve výše uvedených řešeních nevystupují počáteční podmínky, tj. hodnoty $T(0), T(1), T(2), \ldots$ – ty jsou "skryté" v naší $O()$-notaci. Dále v zápise pro zjednodušení zanedbáváme i necelé části argumentů, které mohou být zaokrouhlené.

Příklad 12.8. *Algoritmus merge-sort pro třídění čísel pracuje zhruba následovně:*

- ∗ *Danou posloupnost n čísel rozdělí na dvě (skoro) poloviny.*
- ∗ *Každou polovinu setřídí zvlášť za použití rekurentní aplikace merge-sort.*
- ∗ *Tyto dvě už setříděné poloviny "slije" (anglicky merge) do jedné setříděné výsledné posloupnosti.*

Jaký je celkový počet jeho kroků?

Nechť na vstupu je n čísel. Při rozdělení na poloviny nám vzniknou podproblémy o velikostech $\lceil n/2 \rceil$ a $\lfloor n/2 \rfloor$ (pozor na necelé poloviny). Pokud počet kroků výpočtu označíme $T(n)$, pak rekurzivní volání trvají celkem

$$T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)\,.$$

Dále potřebujeme $c \cdot n$ kroků (kde c je vhodná konstanta) na slití obou částí do výsledného setříděného pole. Celkem tedy vyjde

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn \leq T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil) + cn$$

a to už je tvar řešený v Lematu 12.6 pro $a_1 = a_2 = \frac{1}{2}$. Výsledek tedy je $T(n) = O(n \cdot \log n)$.

(Stejný výsledek by bylo možno získat i z Lematu 12.7 pro $a = b = 2$.) $\square$

Rozšiřující studium

....