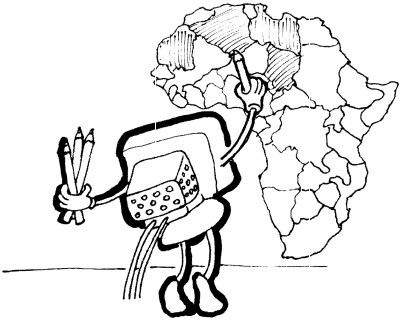
INFORMATIKA

Matematika a počítačové důkazy

ANTONÍM JANČAŘÍK

Pedagogická fakulta UK, Praha



Dvacáté, dnes již minulé století, bylo stoletím vzniku počítačů. Počítače byly vytvořeny v polovině dvacátého století po více než stoleté práci matematiků, která byla zahájena počátkem devatenáctého století *Charlesem Babbagem* a vyvrcholila pracemi zakladatelů teoretické informatiky. Matematici *Alan Turing*, *John von Neumann* a *Lila Kari* vytvářeli abstraktní modely počítačů. *Alonzo Church*, *Moses Schönfinkel*, *Noam Chomsky*, *Emil Post* a *Stephen Cole Kleene* jako první začali pracovat s pojmy z oblasti formální lingvistiky a teorie gramatik.

Většina lidských vynálezů rozšiřovala možnosti lidského těla, počítače rozšiřují možnosti lidského ducha. Počítači byla obohacena i matematika. Počítače stály u zrodu nových aplikovaných oborů matematiky, jakými jsou numerická analýza, optimalizace, statistika, matematická ekonomie či finanční matematika. S rostoucím výkonem počítačů se rozšiřovaly i možnosti jejich využití v matematice. Koncem šedesátých a počátkem sedmdesátých let 20. stol. začaly být počítače využívány pro dokazovaní matematických vět ([1],[2]). Cílem tohoto článku je představit některé výsledky těchto snah a možnosti, které nám počítače při dokazování nabízejí.

Problém čtyř barev

Problém čtyř barev patří mezi nejznámější problémy devatenáctého a dvacátého století. Tento problém byl po takřka stoletém úsilí uzavřen

díky výpočetní technice. Původním autorem problému čtyř barev je *Francis Guthrie*. Problém se stal známým díky jeho bratrovi *Fredericku Guthriemu* a především *De Morganovi*. De Morgan, který řešení problému nenalezl, zaslal problém dalším matematikům (např. i *Hamiltonovi*), z nichž někteří věnovali pokusům o řešení několik let.

Zadání problému je jednoduché: „Lze libovolný rovinný obrazec rozdělený na části obarvit čtyřmi barvami tak, aby každá z částí byla obarvena jednou z barev a aby se na žádné společné hranici částí nevyskytovaly dvě stejné barvy?“

Z vědců, kteří se problémem čtyř barev aktivně zabývali, je nejznámější *Alfred Bray Kempe*, který v roce 1879 v časopise Nature publikoval konstruktivní důkaz, známý jako metoda Kempeho řetězců. Ovšem v roce 1890 dokázal *Percy John Heawood* ve článku nazvaném „Map colouring theorem“, že Kempův důkaz je chybný.

Přestože na problému pracovalo mnoho talentovaných matematiků, nepodařilo se až do roku 1976, tedy po celých 86 let, najít správný a především kompletní důkaz.

V roce 1976 *Appel* a *Haken* za asistence *Kocha* spustili na počítači výpočet, který zhruba po 1200 hodinách strojového času uzavřel jednu etapu matematiky. Program, který napsali, prověřil 1496 konfigurací grafů s nejvýše 14 hranami, které ještě podle jejich teorie reducibility zbývalo prověřit. Tento výpočet přesahoval lidské možnosti a bez využití výpočetní techniky by nebyl možný.

Problém čtyř barev se tak stal prvním velkým matematickým problémem, jehož řešení bylo dosaženo pomocí počítače a nemohlo být přímo ověřeno jinými matematiky. Detaily důkazu byly publikovány ve dvou článcích v roce 1977. Kompletní důkaz a historii problému lze nalézt v [3] a [4].

Dopočet jako dokončení důkazu

Použití počítačů pro řešení problému čtyř barev má ryze technický charakter. Úkolem počítače, resp. počítačového programu, je provést kontrolu v konečném počtu zbývajících případů. V případě problému čtyř barev se jednalo o 1496 možností, které zbývalo prověřit. V této podobě byl počítač použit i pro důkaz některých dalších tvrzení. Další známý případ použití počítačů pochází z teorie samoopravujících kódů. Samoupravující kódy se používají pro odstraňování chyb u přenosu dat zatíženého šumem (více viz např. [5]). Pomocí těchto kódů lze nejen zjistit, zda v přenosu dat došlo k chybě v důsledku šumu, ale i takovou chybu, pokud není příliš velká,

odstranit. Jako příklad samoupravujícího kódu lze uvést kód *Hammingův* (7,4), který získáme tak, že do matice napíšeme všechna čísla od jedné do sedmi ve dvojkové soustavě:

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Hammingův kód tvoří sedmimístné vektory tvořené nulami a jedničkami, které při vynásobení s takto získanou maticí dávají při počítání v modulární aritmetice o základu dva nulový vektor (místo čísla bereme jeho zbytek po dělení dvěma).

Hammingův kód dokáže opravovat jednu chybu. Postup při detekci a opravě chyb je navíc velice jednoduchý. Sedmimístný vektor, který obdržíme z kanálu zatíženého chybou, vynásobíme maticí H. Pokud vyjde nulový vektor, jedná se přímo o odeslané slovo. Pokud je výsledek nenulový, opravíme obdržený vektor na místě, na kterém se výsledek nachází v matici H. Např. vektor (1,1,1,1,0,0,1,1) dává po vynásobení s maticí H vektor (1,1,0) a ten se v matici H nachází ve třetím sloupci. Odesílané slovo tedy bylo (1,1,0,1,0,0,1,1).

Hammingův kód má ještě jednu velice specifickou vlastnost: každý vektor tvořený nulami a jedničkami je buď kódovým slovem nebo se od právě jednoho kódového slova neliší na více místech, než je počet opravovaných chyb. Kódy, které splňují tuto vlastnost, se nazývají perfektní. Matematiky a informatiky pochopitelně zajímalo, kolik takových kódů existuje. Kromě Hammingových kódů byl známým perfektním kódem ještě kód triviální (má pouze dvě kódová slova, tvořená buď samými jedničkami, nebo samými nulami) a dva kódy *Golayovy*. Pomocí nejrůznějších úvah se podařilo ukázat, že nemohou existovat „příliš velké“ perfektní kódy. Poslední krok v důkazu, že jiné než výše zmíněné perfektní kódy neexistují, však nakonec udělaly opět počítače, když prověřily několik tisíc „malých“ zbývajících možností.

Algoritmy a algoritmické myšlení

V druhé polovině dvacátého století začaly počítače ovlivňovat nejrůznější oblasti lidské činnosti. Algoritmické přístupy samozřejmě ovlivnily i matematiku a didaktiku matematiky. Například to byl výzkum automatického dokazování v geometrii, který prováděl *prof. Landa* ([6]). Cílem

jeho práce bylo formulovat určitý předpis, co a jak provádět, aby byla věta dokázána, a vést tak žáky k dokazování základních vět cílevědomě a účelně. Ve stejné době, tedy v padesátých letech minulého století, probíhal podobně orientovaný výzkum i v USA ([7]). Výstupem z tohoto výzkumu však neměl být návod pro žáka, jak postupovat při důkazu, ale program na dokazování logických a geometrických vět pro samočinné počítače.

Jako vhodné pole pro nasazení počítačů při důkazech se nakonec ukázala logika, především predikátová logika prvního řádu, které se budeme v tomto článku dále věnovat. O možnostech využití počítačů pro dokazování vět v geometrii se mohli čtenáři toho časopisu dozvědět z článků [8],[9].

V predikátové logice prvního řádu je důkaz posloupnost tvrzení, které dostáváme postupně jedno z druhého pomocí pevně daných pravidel. Základní myšlenkou automatického dokazování je nechat počítač aplikovat odvozovací pravidla na množinu výchozích tvrzení a tak odvozovat stále nová a nová tvrzení. Tento proces je deterministický, a proto snadno pomocí počítačů realizovatelný. Naprostá většina takto získaných tvrzení je však prakticky neupotřebitelná. Jedná se zpravidla o triviální důsledky výchozích tvrzení získané např. přidáním dalších proměnných. Naším cílem není generovat všechna tvrzení, která se dají v pevně daném počtu kroků získat z výchozích axiomů, ale najít konkrétní cestu z výchozích tvrzení do požadovaného tvrzení získanou tak, že v každém kroku použijeme jedno konkrétní odvozovací pravidlo na jedno nebo více z již známých pravdivých tvrzení. Můžeme tedy nechat počítač generovat pravdivá tvrzení a zároveň kontrolovat, zda námi požadované tvrzení již není mezi dokázanými.

Tento postup by byl při procházení všech možností velmi pomalý. Důkazy mající více než 10 kroků by se již pomalu stávaly nedosažitelnými v rozumném čase. Proto jsou při hledání důkazů pomocí počítačů využívány heuristické algoritmy. Tyto algoritmy se snaží při dokazování „volit“ vhodné cesty k důkazu. Činnost těchto algoritmů lze ovlivňovat pomocí nejrůznějších parametrů. Například lze omezovat počet proměnných, které bude počítač používat, nebo upřednostnit některá odvozovací pravidla před jinými. Nevýhodou heuristických algoritmů je to, že u nich není zaručena stoprocentní úspěšnost. Výpočet může skončit neúspěchem i v případě, že důkaz existuje a procházením všech možností by jej bylo možné nalézt. Přes tento nedostatek se však heuristické algoritmy ukazují jako velice efektivní, a v mnoha případech přinášejí velmi kvalitní výsledky. Otázkou však bylo, zda počítače mohou přinést opravdu nový

a originální výsledek v oblasti, na kterou se matematici delší dobu soustředili. Odpovědí na tuto otázku se stalo vyřešení *Robbinsovy* hypotézy výhradně pomocí počítačové techniky.

Robbinsův axiom

Mezi nejznámější problémy, dokázané prostřednictví výpočetní techniky, patří Robbinsova hypotéza týkající se Booleových algeber. Booleovy algebry jsou algebraickým zobecněním dvou známých struktur.

První z nich je množina všech podmnožin dané množiny s operacemi odpovídajícími průniku, sjednocení a doplňku do množiny S. Nejmenším prvkem 0 v této struktuře je prázdná množina a největším prvkem 1 je celá množina.

Druhou je algebra výroků v dvouhodnotové logice je A = {nepravda, pravda}, 0 = nepravda, 1 = pravda, s operacemi odpovídajícími disjunkci, konjunkci a negaci.

V roce 1933 *E. V. Huntington* definoval Booleovy algebry (viz [10] a [11]) jako strukturu, ve které je operace sčítání komutativní a asociativní a pro operace sčítání a negace je splněno, že $(\neg(\neg x + y) + \neg(\neg x + \neg y)) = x$ pro všechna x, y.

Krátce nato vyslovil *H. Robbins* domněnku, že třetí z axiomů v Huntingtonově definici může být nahrazen axiomem $\neg(\neg x + y) + \neg(\neg x + \neg y) = x$. Vznikly tak dvě definice algeber – Booleovské a Robbinsovy. Je poměrně jednoduché ukázat, že Booleovské algebry jsou Robbinsovy, opačnou implikaci se Huntingtonovi ani Robbinsovi dokázat nepodařilo. Touto otázkou se později zabývalo více matematiků, kromě jiných i *Tarski* se svými žáky (viz [12]).

Od počátku osmdesátých let byly do pokusů dokázat ekvivalenci obou definic zapojeny i programy pro automatické dokazování. Během prvních pokusů se řešení nalézt nepodařilo, bylo však dosaženo dílčích výsledků, které v roce 1996 přispěly k vyřešení problému (např. [13], [14]). Problém byl nakonec vyřešen za pomocí programu EQP. Ten potvrdil, že všechny Robbinsovy algebry jsou Booleovské. Tento výpočet byl posléze zopakován i dalšími programy, které kromě samotné odpovědi vracejí důkaz tvrzení ve formě ověřitelné nezávislými programy (proof checking programs). Důkaz byl publikován v roce 1997 a vzbudil zájem jak mezi odbornou, tak mezi laickou veřejností ([15],[16]).

Komutant

Příklad Robbinsova axiomu ukazuje, že automatické důkazy mohou být mocným nástrojem při dokazování matematických tvrzení. Automatické důkazy mají i svá omezení. Existují například jednoduchá tvrzení, se kterými se seznamují studenti již v základním kurzu algebry, která činí počítačům velké problémy. Jedním z těchto tvrzení je věta, že komutant grupy G tvoří její normální podgrupu. Klasický důkaz totiž není formulován pomocí odvozovacích pravidel, ale využívá vlastností zobrazení, se kterými lze v predikátové logice pracovat jen obtížně – konkrétně vhodného zobrazení a tvrzení, že jádro zobrazení tvoří normální podgrupu. V tomto konkrétním případě se sice nakonec podařilo užitím počítačů sestavit důkaz pomocí odvozování z axiómů, tento důkaz je však výrazně delší a komplikovanější než důkaz klasický.

MV-algebry

Na závěr bych rád ukázal, jakým způsobem programy pro automatické dokazování fungují. Jako ukázku zvolím tvrzení, že basic algebra je MV-algebrou, právě tehdy, když je BCC-algebrou. Jeho autory jsou *I. Chajda* a *R. Halaš* ([17]). Tvrzení bylo prezentované na semináři katedry matematiky PřF UP v Olomouci.

V této chvíli není důležité, jaký je význam jednotlivých pojmů, a soustředíme se pouze na technickou stránku celého problému. Každá z uvedených algeber je strukturou s jednou binární a jednou unární operací, které splňují pevně dané podmínky. Důkaz citovaného tvrzení spočívá v odvození, že z jedné skupiny axiomů lze vyvodit jiné tvrzení. Vzhledem k tomu, že všechna tvrzení lze vyjádřit v predikátové logice prvního řádu, pokusil jsem se pro důkaz tvrzení využít programu Otter. Program Otter patří mezi nejznámější programy pro automatické dokazovaní a je dostupný bezplatně na síti internet. Informaci o programu Otter i dalších programech a možnostech jejich využívání lze nalézt na adrese [18].

Program Otter pracuje v DOSovém režimu a spouští se pomocí dávkového souboru. Pro běh programu v základním režimu jsou potřeba dva parametry – jméno vstupního a jméno výstupního souboru. Do vstupního souboru se zadávají axiomy, které se mají dokázat. V našem případě jsou to axiomy basic algebry:

```
x=x.
(x,n)=x.
```

```
m(m(x))=x.
p(x,m(n))=m(n).
p(m(n),x)=m(n).
p(m(p(m(x),y)),y)=p(m(p(m(y),x)),  x).
p(m(p(m(p(x,y)),y)),y)=p(x,y).
p(m(p(m(p(m(p(x,y)),y)),z)),p(x,z ))=m(n).
```

Jeden z ekvivalentních axiomů BCC-algebry:

```
p(m(p(x,y)),p(m(p(z,m(x))),p(z,y)))=m(n).
```

Všechny operace se píší v prefixové formě, která je při zadávání dat do počítače běžné (např. `p(m(p(m(p(x,y)),y)),y)=p(x,y)` je zápis tvrzení $\neg(\neg(x \otimes y) \otimes y) \otimes y = x \otimes y$. Více o jednotlivých formách zápisu a důvodech jejich použití lze nalézt v [19], [20].

Do vstupního souboru dále zadáváme negaci tvrzení, které chceme dokázat. V našem případě potřebujeme dokázat, že z uvedených axiomů vyplývá asociativita:

```
p(p(a,b),c)!=p(a,p(b,c)).
```

Do vstupního soboru je možné zadat také další parametry, které ovlivňují průběh výpočtu.

Po dokončení výpočtu jsou všechny informace o průběhu výpočtu zapsané ve výstupním souboru. Ve výše uvedeném případě trval výpočet 17 sekund, během nichž počítač prošel cca 5000 výrazů a našel důkaz asociativity vyplývající z uvedených axiomů skládající se z 16 kroků:

```
Length of proof is 16. Level of proof is 6.
---------------- PROOF ----------------
1 [] p(p(a,b),c)!=p(a,p(b,c)).
4,3 [] p(x,n)=x.
6,5 [] m(m(x))=x.
8,7 [] p(x,m(n))=m(n).
9 [] p(m(n),x)=m(n).
11 [] p(m(p(m(x),y)),y)=p(m(p(m(y),x)),x).
14 [] p(m(p(m(p(m(p(x,y)),y)),z)),p(x,z))=m(n).
16 [] p(m(p(x,y)),p(m(p(z,m(x))),p(z,y)))=m(n).
18 [para_into,11.1.1.1.1.1,5.1.1]
p(m(p(x,y)),y)=p(m(p(m(y),m(x))),m(x)).
```

```
22,21 [para_into,11.1.1.1.1.1,9.1.1,demod,6,4,6,4] p(n,x)=x.
23 [para_into,11.1.1.1.1.1,7.1.1,demod,6,8,6,22]
m(n)=p(m(x),x).
32,31 [para_from,23.1.1,5.1.1.1] m(p(m(x),x))=n.
71 [para_into,14.1.1.2,11.1.1]
p(m(p(m(p(m(p(m(p(m(x),y)),z)),z)),z)),y)),
p(m(p(m(y),x)),x))=m(n).
104 [para_into,16.1.1.1.1.1,3.1.1,demod,4]
p(m(x),p(m(p(y,m(x))),y))=m(n).
125 [para_from,16.1.1,14.1.1.1.1.1,demod,6,6,22]
p(x,p(m(p(y,p(x,z))),p(y,z)))=m(n).
511 [para_from,71.1.1,18.1.1.1.1.1,demod,6,22,6,6,flip.1]
p(m(p(m(p(m(p(m(x),y)),y)),
p(m(p(m(p(m(p(m(y),x)),z)),z)),x))),
p(m(p(m(p(m(p(m(y),x)),z)),z)),x))=p(m(p(m(x),y)),y).
1003 [para_into,104.1.1.1,5.1.1,demod,6]
p(x,p(m(p(y,x)),y))=m(n).
1449,1448 [para_into,125.1.1.2.1.1,125.1.1,demod,6,22]
p(m(p(x,p(y,z))),p(y,p(x,z)))=m(n).
1687,1686 [para_from,1003.1.1,125.1.1.2.1.1,demod,6,22]
p(m(p(x,y)),p(y,x))=m(n).
2590 [para_from,1686.1.1,18.1.1.1.1.1,demod,6,22,6,1687,6,6,22]
p(x,y)=p(y,x).
2786 [para_from,2590.1.1,1.1.1] p(c,p(a,b))!=p(a,p(b,c)).
4986[para_from,1448.1.1,511.1.1.2.1.1.1.1.1.1,demod,
1449,6,22,1449,
6,22,32,22,1449,6,6,22, 32,22,22,1449,6,22]
p(x,p(y,z))=p(y,p(x,z)).
4987 [para_into,4986.1.1.2,2590.1.1] p(x,p(y,z))=p(z,p(x,y)).
4990 [copy,4987,flip.1] p(x,p(y,z))=p(y,p(z,x)).
4991 [binary,4990.1,2786.1] $F.
------------ end of proof -------------
```

Na prvních osmi řádcích důkazu jsou zopakována tvrzení, se kterými počítač v průběhu výpočtu pracuje. Na dalších řádcích je vždy okomentován postup, který počítač použil, a tvrzení, které použitím pravidla obdržel. Vezměme si například první obdržený výraz:

```
p(m(p(x,y)),y)=p(m(p(m(y),m(x))), m(x)).
```

Postup odvození je popsán následovně:

```
[para_into,11.1.1.1.1.1,5.1.1].
```

Toto tvrzení počítač získal tak, že do výrazu z řádku začínajícího číslem 11 dosadil za x negaci x a následně použil pravidlo, že negace negace x je x z řádku začínajícího číslem 5. Obdobným způsob je možné rekonstruovat všechna tvrzení v důkazu. Zajímavostí je, že počítač, stejně jako autoři důkazu, nejprve dokazuje, že nově vzniklá struktura je komutativní (řádek 2590) a teprve s její pomocí dokazuje platnost asociativity.

Závěr

Počítače rozšiřují lidské možnosti. Díky počítačům můžeme nejen provádět dříve nepředstavitelné výpočty, ale i dokazovat tvrzení, se kterými si matematici nevěděli po celá desetiletí rady. Automatické důkazy nikdy nenahradí nápady matematiků. Ti udávají směr, kterým mají počítače hledat. Počítače ale mohou být vhodným nástrojem pro prověřování hypotéz a rychlé kontroly důkazů či konstruování příkladů a protipříkladů. Díky tomu, že většina programů pro automatizované důkazy je volně dostupná, je možné si jejich práci vyzkoušet a představit studentům, například ve výběrových seminářích věnovaných informatice či matematice.

Příspěvek byl vypracován s podporou grantu GAČR 406/05/P561.

Literatura

[1] *Wang, H.*: Toward Machanical Matematice. IBM, J. Res. Develop. 4(1), 1966.
[2] *Gelernter, H.*: Theorem Proving by Machina. Proc. Of the Summer Inst. Of Symb. Logic at Cornel university 1957.
[3] *Fritsch, R. – Fritsch G.*: The Four-Color Theorem: History, Topological Foundations and Idea of Proof, Springer, 1998.
[4] *Appel, K. I. – Haken, W.*: Every planar map is four colorable. AMS, 1989.
[5] *Lint, J.H.*: Introduction to Coding Theory. Springer-Verlag, Berlin, 1992.
[6] *Landa, L. N.*: Algoritmy a učení. SPN Praha, 1973.
[7] *Newell, A.– Shaw, J.C. – Simon, H.A.*: Report on a general problem-solving program. Proceedings of the International Conference on Information Processing, 1959.
[8] *Pech, P.*: Dokazování a objevování vět v geometrii pomocí metod počítačové algebry I. MFI, r. 15 (2005/06) č. 5, s. 295–308.
[9] *Pech, P.*: Dokazování a objevování vět v geometrii pomocí metod počítačové algebry I. MFI, r. 15 (2005/06) č. 5, s. 369–374.

[10] *Huntington, E. V.*: New sets of independent postulates for the algebra of logic. Trans. AMS 35, 274–304, 1933.
[11] *Huntington, E. V.*: Boolean algebra: A correction. Trans. AMS 35, 557–558, 1933.
[12] *Henkin, L. – Monk, J. D. – Tarski, A.*: Cylindric Algebra. Part I, North–Holland, 1971.
[13] *Winker, S.*: Robbins Algebra: Conditions That Make a Near–Boolean Algebra Boolean. J. Automated Reasoning 6(4), 465–489, 1990.
[14] *Winker, S.*: Absorption and idempotency criteria for a problem in near-Boolean algebra. J. Algebra 153(2), 414–423, 1992.
[15] *McCune, W.*: Solution of the Robbins Problem, J. Automated Reasoning 19(3), 263–276, 1997.
[16] *Kolata, G.*: Computer Math Proof Shows Reasoning Power [online] c 1997. The New York Times Copany. http://www.nytimes.com/librery/cybr/week/1210math.html
[17] *Chajda, I. – Halaš, R.*: A basic algebra is an MV-algebra if and only if is a BCC-algebra. International Jurnal for Theoretical Physics (Přijato k tisku).
[18] *Wiedijk, F.*: [online]. http://www.cs.ru.nl/~freek/digimath/xbylogicx.html
[19] *Habiballa, H. – Volná, E. – Fojtík, R.*: Od teorie formálních jazyků k jednoduchému překladači I. MFI, r. 17 (2007/08) č. 3, s. 171–182.
[20] *Habiballa, H. – Volná, E. – Fojtík, R.*: Od teorie formálních jazyků k jednoduchému překladači I. MFI, r. 17 (2007/08) č. 4, s. 241–244.

Bod v trojúhelníku

STANISLAV TRÁVNÍČEK

Přírodovědecká fakulta UP, Olomouc

Budeme se zabývat úlohou, v níž studentům asi nebude dělat problémy napsat program, který ji řeší, jako spíše matematická analýza problému. I když není příliš obtížná, může studentům chvíli trvat, než nějaký vhodný algoritmus řešení objeví.

Úloha: Reálnými souřadnicemi jsou dány 3 body A, B, C v rovině tvořící (zpravidla) trojúhelník ABC a ještě 4. bod D. Máme rozhodnout, zda bod D je bodem trojúhelníku ABC (tj. zda leží uvnitř nebo na hranici trojúhelníku).

Analýza: Nosná myšlenka je tato: Leží-li bod D v trojúhelníku ABC, pak bod D leží v téže polorovině s hraniční přímkou AB jako bod C a podobně

pro další dvě strany trojúhelníku, tedy nakonec: bod D je bodem trojúhelníku ABC, právě když bod D leží v téže polorovině s hraniční přímkou AB (BC, CA) jako bod C (A, B), tj. platí-li současně všechny tyto tři vztahy.

Nyní je třeba převést tuto myšlenku do analytického vyjádření. Nechť $A[x_1, y_1]$, $B[x_2, y_2]$; nejprve je třeba nalézt obecnou rovnici přímky dané těmito dvěma body. To již bylo provedeno v [1] a odsud plyne, že tato rovnice je:

$$(y_1 - y_2)x + (x_2 - x_1)y + (x_1 y_2 - x_2 y_1) = 0 \qquad (1)$$

Označme

$$U(i, j, k) = (y_i - y_j)x_k + (x_j - x_i)y_k + (x_i y_j - x_j y_i).$$

Dosadíme-li do levé strany rovnice (1) bod $C[x_3, y_3]$, dostaneme číslo $U(1, 2, 3)$, dosadíme-li do levé strany rovnice (1) bod $D[x_4, y_4]$, dostaneme číslo $U(1, 2, 4)$. Tedy: leží-li bod D v téže polorovině s hraniční přímkou AB jako bod C, jsou čísla $U(1, 2, 3)$, $U(1, 2, 4)$ buď obě nezáporná nebo obě nekladná, tedy $U(1, 2, 3) \cdot U(1, 2, 4) \geq 0$. Přesné hodnoty těchto dvou čísel nás totiž ani nezajímají, jen jejich znaménka. Další dva případy (tj. při hraniční přímce BC a CA) nám doplňují trojici podmínek, která tvoří nutnou a postačující podmínku toho, aby bod D byl bodem trojúhelníku ABC:

$$U(1, 2, 3) \cdot U(1, 2, 4) \geq 0,$$
$$U(2, 3, 1) \cdot U(2, 3, 4) \geq 0,$$
$$U(3, 1, 2) \cdot U(3, 1, 4) \geq 0.$$

Následující program *Body4a* je vytvořen podle této analýzy.

```
program Body4a;

var
  I: Integer;
  X, Y: array [1..4] of Real;
  TrojZadan: Boolean;

function U(P, Q, R: Integer): Real;
```

```
begin {U}
  U := (Y[P]-Y[Q])*X[R]+
       (X[Q]-X[P])*Y[R]+
       X[P]*Y[Q]-X[Q]*Y[P]
end;  {U}

begin {program}
  WriteLn;
  repeat
    I := 1; TrojZadan := true;
    WriteLn('Zadej vrcholy trojuhelniku: ');
    repeat
      Write('x', I, ': ');
      ReadLn(X[I]);
      Write('y', I, ': ');
      ReadLn(Y[I]);
      Inc(I)
    until I = 4;
    if U(1, 2, 3) = 0 then
    begin
      WriteLn('Nebyl zadan trojuhelnik');
      TrojZadan := false
    end
  until TrojZadan;
  WriteLn('Zadej 4. bod: ');
  Write('x: ');
  ReadLn(X[4]);
  Write('y: ');
  ReadLn(Y[4]);
  if (U(1, 2, 3) * U(1, 2, 4) >= 0) and
     (U(2, 3, 1) * U(2, 3, 4) >= 0) and
     (U(3, 1, 2) * U(3, 1, 4) >= 0)
    then
      WriteLn('Tento bod je v trojuhelniku')
  else
      WriteLn('Tento bod je vne trojuhelniku');
  ReadLn
end.  {program}
```

Jiné řešení. Nosná myšlenka: Bod D je bodem trojúhelníku ABC, právě když pro obsahy trojúhelníků platí $P(ABD) + P(BCD) + P(CAD) = P(ABC)$.

Označme vektory $\boldsymbol{u} = B - A$, $\boldsymbol{v} = C - A$ a uvažujme vektorový součin

$$\boldsymbol{u}\times\boldsymbol{v} = \begin{pmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ x_2 - x_1 & y_2 - y_1 & 0 \\ x_3 - x_1 & y_3 - y_1 & 0 \end{pmatrix} = ((x_2-x_1)(y_3-y_1)-(x_3-x_1)(y_2-y_1))\boldsymbol{k}.$$

Víme, že $|\boldsymbol{u}\times\boldsymbol{v}| = |\boldsymbol{u}|\cdot|\boldsymbol{v}|\cdot\sin\varphi$, kde je úhel φ těchto vektorů, tedy vnitřní úhel při vrcholu A v trojúhelníku ABC; $|\boldsymbol{u} \times \boldsymbol{v}|$ tedy udává dvojnásobný obsah trojúhelníku ABC. Platí tak $2P(ABC) = |\boldsymbol{u} \times \boldsymbol{v}| = |(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$ a po úpravě

$$P(ABC) = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)/2. \qquad (2)$$

(Všimněme si, že v absolutní hodnotě je tu vlastně stejný výraz, jako je na levé straně rovnice (1).) Stejně lze vyjádřit i obsahy zbývajících tří trojúhelníků $P(ABD)$, $P(BCD)$ a $P(CAD)$.

Uveďme si nyní program *Body4b* vytvořený studentem gymnázia a založený na tomto přístupu.

```
program Body4b;

var
  X1, Y1, X2, Y2, X3, Y3, X, Y: Real;

function Test: Boolean;
var
  P, P1, P2, P3: Real;
begin {Test}
  P := Abs((X1*(Y2-Y3)+X2*(Y3-Y1)+
       X3*(Y1-Y2))/2);
  P1 :=Abs((X1*(Y2-Y)+X2*(Y-Y1)+
       X*(Y1-Y2))/2);
  P2 :=Abs((X1*(Y-Y3)+X*(Y3-Y1)+
       X3*(Y1-Y))/2);
  P3 :=Abs((X*(Y2-Y3)+X2*(Y3-Y)+
```

```
       X3*(Y-Y2))/2);
  Test := (P = P1+P2+P3)
end;  {Test}

begin {program}
  Write('Zadej souradnicemi 3 body: ');
  ReadLn(X1, Y1, X2, Y2, X3, Y3);
  Write('Zadej 4. bod: ');
  ReadLn(X, Y);
  Write('Tento bod je ');
  if Test then
    WriteLn('v trojuhelniku.')
  else
    WriteLn('vne trojuhelniku.');
  ReadLn
end.  {program}
```

Zadejte úlohu svým studentům nebo je seznamte s uvedenými dvěma programy. V programu *Body4b* se netestuje, zda body ABC tvoří trojúhelník, což je chyba, protože pokud body A, B, C leží na přímce, program reaguje chybně – hlásí, že *každý* bod této přímky „je v trojúhelníku“. Nechť vaši žáci test doplní, případně i program přepracují s tím, že do něj doplní i jistý minimální komfort, jaký je užitý v programu *Body4a*.

Literatura

[1] *Trávníček, S.*: Přímka daná dvěma body. MFI 17 (2007 – 08), č. 7, s. 433 – 439.