Pokroky matematiky, fyziky a astronomie

Jiří Hořejš

Struktura matematiků a programů

Pokroky matematiky, fyziky a astronomie, Vol. 21 (1976), No. 2, 84--88

Persistent URL: http://dml.cz/dmlcz/139255

Terms of use:

© Jednota českých matematiků a fyziků, 1976

Institute of Mathematics of the Academy of Sciences of the Czech Republic provides access to digitized documents strictly for personal use. Each copy of any part of this document must contain these *Terms of use*.

This paper has been digitized, optimized for electronic delivery and stamped with digital signature within the project *DML-CZ: The Czech Digital Mathematics Library* http://project.dml.cz

Struktura matematiků a programů*)

Jiří Hořejš, Brno

V tomto textu „matematik" označuje absolventa klasického matematického vzdělání, kterému jeho zaměstnání umožňuje věnovat se výzkumu v oblasti, v níž nalezl zalíbení. Učitelé matematických kateder vysokých škol mají většinou k tomuto výměru dost blízko.

Prohlásit, že tato třída matematiků dnes představuje jednu z posledních bašt, která se stále ještě úspěšně brání invazi počítačů, by bylo trochu odvážné a nebezpečné. Zeslabíme-li však takové tvrzení vhodnými existenčními kvantifikátory, nebudeme tak daleko od pravdy. A s určitostí platí, že běžná laická představa podporovaná vědecko-fantastickou literaturou, podle níž co matematik to kybernetik obsluhující myslící roboty, se nezakládá na pravdě.

Toto povídání vzniklo ze snahy zjistit, proč stále ještě mnoho matematiků cítí jistou nedůvěru k výpočetní technice a dává přednost tomu žít a pracovat mimo její dosah. A současně ukázat, že některé z příčin spočívají možná jen v neznalosti současných metod práce s počítači.

Ekonom nebo sociální psycholog by mohl přijít na myšlenku, že averze matematiků k počítačům má stejné důvody jako averze tkalců k pletacím stavům někdy v 17. století – že jde o strach z konkurence. Avšak paralela tu zcela zřejmě kulhá. Srovnávat v tomto smyslu průmyslovou revoluci na počátku kapitalismu s obdobím vědeckotechnické revoluce v podmínkách socialismu by mohl nejvýš ten, kdo nemá vůbec představu o stále rostoucích požadavcích na počet absolventů všech středoškolských i vysokoškolských studijních směrů, které mají jen trochu co dělat s výpočetní technikou.

Pokud bychom se zajímali pouze o vnější důvody, museli bychom spíš popřát sluchu poukazu na dost neutěšenou situaci související s formami odborné přípravy a vědeckého růstu. Absolvent matematiky, vážný zájemce o vědeckou práci v oblasti počítačů, narazí skutečně na větší potíže s hledáním vhodného tématu a komise pro obhajoby příslušných hodností než ten, kdo se bude věnovat klasickým disciplínám. Doufejme ale, že i tady přinese blízká budoucnost radikální nápravu a uvažme raději důvody vyplývající z podstaty věci.

Asi každý z nás byl mnohokrát osloven: Ty jsi matematik, přepočítej ty peníze (knihy, lidi . . .). Zasvěcenější laici ukládají ovšem úlohy přiměřenější: sčítat a násobit zpaměti dvojciferná čísla, převádět minuty na hodiny, ba i na trojčlenku často dojde. Zadavatele takových úkolů stíháme oprávněným úsměškem (s příp. stopou strachu z nedůvěry ve vlastní schopnosti) – neví, co to je matematika a matematik. Plete si matematiku s počítáním. A přece se podobné záměny stále ještě dopouští i profesionální matematik: slovo „počítač" ho svádí k představě zařízení sloužícího výhradně k řešení numericky orientovaných úloh. Pan Francis Bacon (když už jsme tu jednou citovali 17. století) by

*) Předneseno autorem na 11. celostátní konferenci o vyučování matematice na VŠTEZ.

zařadil tento omyl mezi „idoly tržiště" a viděl v něm jednu z příčin, proč se od počítačů odklánějí matematici zaměření nenumericky. A při tom je fakt, že dnes převážná většina celosvětového „strojového času" není již věnována číselnému zpracování úloh analýzy, ale zpracování informace zcela obecné povahy: slov a textů zachycujících evidenci a správu, matic zapisujících grafy systémových vazeb, výrazů podrobovaných formálním úpravám v rámci daného algebraického systému či logického kalkulu, datových struktur reprezentujících vícedimenzionální geometrické útvary zobrazované ve zvolené projekci na obrazovce atp.

Pak je tu další důvod. Matematik miluje obecnost, abstrakci. Činí mu potěšení odsouvat nepodstatné detaily (a jako každý, rád za takové považuje i ty, s nimiž si neví rady). Odtud záliba v algebraizaci; nejen matematiky, ale i matematiků. Za klamný „idol kmene" je pak možno považovat přesvědčení, že jakákoliv práce s počítači vykazuje právě opačné vlastnosti, že tu jde především o suchopárnou mravenčí práci bez invence, bez možnosti tvořit stavbu. Nuže, kdo jen jedním okem nahlédl do problematiky současné teoretické informatiky ví, že tuto výtku nelze adresovat jejím disciplínám; teorie jazyků, teorie vyčíslitelnosti, teorie složitosti — to vše jsou krásné, matematicky elegantní teorie; navíc mají občas zcela bezprostřední aplikace (u mnoha teorií jiných matematických disciplín je třeba značnou dávku abstrakce a myšlenkového úsilí věnovat na to, aby proklamované aplikace bylo aspoň zdáli vidět). Teoretická informatika ovšem není programování a je to samozřejmě právě programování, které je nejčastěji napadáno. Jedni obviňují programování z toho, že je to řemeslo, druzí z toho, že je to umění. V žádném případě ne věda, neřku-li matematika. Je třeba uznat, že většina programů je tvořena z externích popudů, a tedy míra tvořivosti je předem omezena. Je dále třeba přiznat, že v pionýrských dobách byla práce programátora soustředěna na „technologické" otázky konstrukce jednotlivých příkazů víc než na sám řešený problém. Zatímco však odpověď na první námitku je teoreticky jednoduchá (vymýšlejme zadání sami — i ve vlastní vědecké práci v „abstraktní" matematice se mohou počítače uplatnit), proces konverze programovacích technik v samostatnou metodologii návrhu algoritmizace a programování se právě dostává na scénu celosvětového zájmu (i) matematiků.

Snaha po systematickém programování umožní jistě prohlásit za nepravý i další dojem, který řada praktiků má a které by p. Bacon zařadil nejspíš mezi „idoly divadla": totiž představu, že výsledky programátorovy práce — programy — jsou téměř esoterická díla, srozumitelná nejvýš jemu samému (a po určité době od jejich stvoření ani jemu již ne). Skutečně, špatně členěný program představuje natolik složitý a nepřehledný systém detailů, že míra možnosti komunikace, po níž každý matematik svou přirozeností touží (na rozdíl od některých představitelů řady jiných profesí, kteří z konkurenčních důvodů raději výsledky své práce tají), je mizivá nebo nákladná.

Zdá se tedy, že neškodí připomenout některé novější zásady a pohledy na strukturu a tvorbu programů; měly by přispět k odstranění šedivých brýlí, skrz něž někdy matematik na programování pohlíží.

Každý úkol, s nímž zamýšlíme jít na počítač, se dá obecně charakterizovat pojmem, který je matematikovi vlastní: v počítači chceme totiž vždy získat zařízení realizující *funkci* F, jež k předloženým *vstupním datům* X nalezne žádaná *výstupní data* Y, $Y = F(X)$. Jakákoliv (jakkoliv abstraktní a třeba i nekonstruktivní) definice funkce F definuje

současně jistý *fiktivní počítač*; jazyk, ve kterém jsme schopni tuto funkci popsat, je jeho *jazykem příkazů* a podobně zní i výměr *jazyka dat*. Mít k dispozici tento počítač, bylo by po starosti a funkční hodnota by byla dosažitelná pro všechna X z definičního oboru funkce F (mimochodem, počítač velmi dobře konkretizuje a precizuje řadu obtížných a někdy i filozoficky problematických pojmů souvisejících s vymezením definičních oborů, rozdílů mezi proměnnou a její hodnotou atp.).

Problém je v tom, že ne ke každé úloze je k dispozici (jednoúčelový) počítač pro její řešení. Jazyk příkazů i dat zvolené úlohy je na to obvykle příliš mocný. Za to bývá k dispozici *skutečný počítač* (ať již jeho jazykem je „jazyk stroje" nebo některý z obecných programovacích jazyků); jeho výhodou je, že existuje, jeho nevýhodou skutečnost, že příkazy a data, které zvládá jsou podstatně jednodušší, než se nám hodí. Most mezi snem a realitou tu však je — je to *strukturace*. U skutečného počítače jsou vedle množin elementárních příkazů a dat dána *pravidla kompozice*, umožňující vytvářet složitější celky. U fiktivních počítačů nebrání teoreticky nic tomu, aby si jeho tvůrce vymyslil i *pravidla rozkladu* do složek a užil zásady divide et impera na příkazy i data. A tu jsme téměř u jádra věci: zatímco klasické metody programování vycházely od skutečného počítače k fiktivnímu, *zdola nahoru*, dnes získává stále více popularity postup opačný, *shora dolů*. Metody zdola nahoru zcela přirozeně preferují prostředky (tj. syntaxi a sémantiku jazyka skutečného počítače) před cíli (tj. sémantikou fiktivního počítače — úlohy), programování shora dolů lépe umožňuje naopak myslet problémově, k věci.

Při řešení obtížných úloh je „vzdálenost" mezi fiktivním a skutečným počítačem značná a přechod od prvního k druhému (při předpokládaném postupu shora dolů) je třeba vytvářet postupně, víceetapově. Každá etapa rozkládá příkazy a data do jednodušších, vytváří stále ještě fiktivní, ale skutečnosti se přibližující počítače. Aby tento proces postupných rozkladů probíhal bez potíží, je nutno pečlivě volit metody strukturace; rozklad do systému s příliš mnoha vazbami se může zdát být pohodlnější na jedné etapě, téměř určitě však zkomplikuje pokusy o rozklad v etapách dalších. Výběr strukturačních metod je proto pro úspěšnou „elegantní" tvorbu programů podstatný. Teoretické studie i zkušenosti z praxe (a zdůrazněme: nikoliv školské, ale naopak celosvětové zkušenosti s výstavbou velkých programových systémů) navrhují prohlásit některé přirozené způsoby rozkladu za *standardní*, zahrnout jejich vyjádření do užívaných programovacích jazyků a omezit se na ně.

Pokud jde o jazyk příkazů, ukazuje se, že stačí tři způsoby rozkladu; první je matematikovi zcela běžný, druhý je obvyklý, třetí se v klasické matematice užívá méně. Jde o rozklady funkce $Y = F(X)$ na

(*i*) $F(X) = F_2(F_1(X))$

(*ii*) $F(X) = \begin{cases} F_1(X), \text{ jestliže } \alpha\,(X) \\ F_2(X) \text{ v opačném případě} \end{cases}$

(*iii*) $F(X) = F_1^n(X)$, kde $n \geqq 0$ je voleno tak, že $\alpha(F_1^i(X))$ pro všechna i, $0 \leqq i < n$ zatímco *non* $\alpha(F_1^n(X))$.

Zde F_1, F_2 označuje opět funkce, α predikát (funkce nabývající hodnoty pravda — lež); určení definičních oborů ponecháváme čtenářovi. Vyjádření těchto rozkladů jsou

vtělena do většiny programovacích jazyků; (*i*) odpovídá *sekvenci* F_1; F_2, (*ii*) *větvení* **if** α **then** F_1 **else** F_2, (*iii*) *iteraci* **while** α **do** F_1; samozřejmě, setkáváme se i s jinými zápisy téže ideje, někdy šťastnými, jindy méně výstižnými.

Pozoruhodné tu je nikoliv to, co je, ale to, co *není* doporučováno: mezi uvedenými způsoby strukturace není uvedena žádná forma tzv. *skokových příkazů* (**goto**), jež je u většiny programovacích jazyků obvyklá. Boj za přehlednou strukturaci je tedy mj. bojem proti užívání takových příkazů, které jsou zcela cizími elementy z hlediska matematické teorie funkcí, teoreticky a ve zdrcující většině případů i prakticky postradatelné; jejich užití je ve většině případů dáno již zmíněnou nadvládou prostředků nad smyslem díla, vede k složitým a nepřirozeným vazbám, které znemožňují účinnou verifikaci atp.

Také v oblasti dat se ustálil výčet hlavních struktur do několika, pomocí nichž lze (obecně ovšem opět hierarchicky) zachytit data zpracovávaná na jednotlivých fiktivních počítačích. Než některé z nich uvedeme, zdůrazněme, že rozklad dat postupuje souběžně s rozkladem příkazů — u obou entit se globální představy a popisy postupně zaostřují a zpřesňují a složky původně jednolitých příkazů pracují na další etapě obvykle nad složkami původně jednolitých dat.

Na data nazíráme jako na hodnoty jistých *proměnných*; množina hodnot, kterých může proměnná nabýt, udává její *typ*. Každá proměnná má své *jméno*, pomocí něhož se na ni odvoláváme; programovací jazyk má prostředky (opět: někdy vhodné, jinde méně výstižné), jak *deklarovat* proměnnou, tj. určit její jméno a typ. I pro proměnné uvedeme pouze tři způsoby strukturace do složek; na složky se pak odvoláváme pomocí jména proměnné a vhodného *selektoru*. (*i*) *pole* je proměnná strukturovaná do předem stanoveného počtu složek téhož typu; selektorem jsou *indexy* — matematika zná pojem dvourozměrného pole pod jménem matice; (*ii*) *soubor* je proměnná strukturovaná do neurčeného počtu lineárně uspořádaných složek téhož typu; selektorem je *pozice* udávající (v každém okamžiku jedinou) přístupnou složku; (*iii*) *záznam* je proměnná strukturovaná do předem stanoveného počtu složek obecně různého typu; složky se tu nazývají *položky* a jsou opatřeny také jmény, která fungují jako selektory.

Načrtli jsme prostředky pro postupný rozklad problému; proces rozkládání končí, jakmile se octneme u atomických příkazů a dat skutečného počítače; u počítačů, jejichž jazykem je některý z běžných obecných programovacích jazyků (PASCAL, ALGOL, COBOL, FORTRAN aj., přičemž pořadí zde uvedené přibližně udává relativní míru souladu jazyka se zásadami tu uvedenými), budou mezi atomická data patřit proměnné typu **real, integer, string,** ... (nabývající jako hodnot po řadě reálných čísel, celých čísel, textových řetězců ...) a mezi příkazy atomické příkazy přiřazující takové proměnné hodnotu výrazu sestaveného z jiných atomických proměnných a atomických funkcí (operací) zadávajících přirozeným způsobem aritmetiku nad hodnotami proměnných příslušných typů.

Není tu bohužel možno zabývat se detailněji blahodárnými vlivy, které strukturovaná tvorba programů má na možnosti verifikace programů (od obyčejného ladění až po formální důkazy v rámci predikátového počtu), jejich přehlednost, modifikovatelnost, portabilitu atp., a proto omezený rozsah příspěvku a autorových vyjadřovacích pro-

středků nejspíš nepřesvědčí toho, kdo nechce být přesvědčen (o tom, že programování se pomalu stává vědecky fundovanou metodikou hodnou pozornosti matematiků). Případný zájemce však najde další argumenty a zejména detailnější výklad v uvedené literatuře, kde tituly [1] a [2] jsou zpracovány světovými kapacitami s bohatou teoretickou erudicí i praktickými zkušenostmi; jediná šance článku [3] naproti tomu spočívá v jeho relativně větší dostupnosti.

Literatura

[1] O. J. Dahl, E. W. Dijkstra, C. A. R. Hoare: *Structured Programming*, Academic Press 1972.
[2] D. E. Knuth: *Structured Programming with Goto Statements*, Computing Surveys 1974, Nr. 4, p. 261—301.
[3] J. Hořejš: *Principy strukturovaného programování I, II*, Informačné systémy 1975, č. 2, str. 97—115, č. 3, str. 227—244.
[4] F. Bacon: *Nové Organon*, ČSAVU 1922.

Computer science a jej vzťah k matematike

Donald E. Knuth, Stanford, USA)*

Súčasný trend umožňuje predvídať deň, keď computer science a matematika budú obe existovať ako uctievané disciplíny slúžiace analogickým, ale rozdielnym úlohám vo vzdelaní človeka.

Nová disciplína, nazývaná computer science, sa objavila v programoch väčšiny svetových univerzít. Tento článok opisuje subjektívny pohľad na vzťah tohoto predmetu k matematike formou porovnania ich rozdielov a podobností a skúmaním niektorých spôsobov, ktorými si navzájom pomáhajú.

Čo je to computer science?

Pretože computer science je pomerne mladá disciplína, musím najskôr vysvetliť, o čom pojednáva. Aspoň moja manželka vraví, že to musí vysvetľovať, kedykoľvek sa jej niekto spýta, čo robím. Navyše mám pocit, že v súčasnosti väčšina ľudí chápe

*) Reprinted from The American Mathematical Monthly, April 1974, copyright © by the Mathematical Association of America. (Zkrácené znění.)