Jiří Hořejš

Teoretické základy informatiky

Pokroky matematiky, fyziky a astronomie, Vol. 20 (1975), No. 1, 15--26

Persistent URL: http://dml.cz/dmlcz/139846

Teoretické základy informatiky

Jiří Hořejš,) Brno*

1.

Hovoříme-li o *informatice*, máme na mysli soustavu vědních disciplín zabývajících se obecným výzkumem informačních struktur a procesů a všímajících si počítačů jakožto základního a univerzálního umělého prostředku pro zpracování informace. Toto rámcové vymezení náplně informatiky neurčuje ovšem jednoznačně její hranice a je třeba počítat s tím, že vzhledem k jejímu charakteru a šíři bude kolísat zdůrazňování těch či oněch jejích součástí podle zaměření jednotlivých pracovišť.

Specializace bude pak určena zvláště tím, zda dané pracoviště (speciálně např. v rámci vysokoškolské výuky) se bude soustřeďovat především na obecné teoretické aspekty informatiky (*teoretická informatika*) nebo spíš na problematiku související s počítači (*počítačová* neboli *konkrétní informatika*); ve druhém ze jmenovaných případů lze počítat s další diferenciací podle toho, zda hlavní náplní je či není vlastní konstrukce počítačů, resp. jejich programového vybavení, podle zaměření na speciální oblasti aplikací, atd.

Na nahoře naznačený pokus o výměr pojmu informatiky je možno navázat pokusem o výčet disciplín, které konstituují osu její teoretické, resp. počítačové větve. Podívejme se nejprve na obecnou stránku věci.

V základě leží pojem *informace*. S tímto termínem je obvykle spojena představa poznatků o určité problematice, zmenšování nejistoty o skutečnosti, systému výroků či zpráv. O precizaci těchto představ usiluje (matematická) *logika*, *pravděpodobnostní teorie informace*, vědní obory zabývající se *informačními systémy* (dokumentací ap.) i jiné; každá disciplína z jiného hlediska, jinými prostředky a z jiných důvodů. Nás bude zajímat spíš *struktura informace*, její zápis v určitém jazyce a zejména pak proces odvozování nové informace z informace dané (tzv. *zpracování informace*). Pro současnou praxi je nejdůležitější případ, kdy zpracování informace probíhá podle jistého návodu, algoritmu (jeho zápis v jistém jazyce se nazývá program), v určitém abstraktním či konkrétním systému; termíny procesor, automat, paměť aj. jsou podle okolností používány k pojmenování různých složek takového systému.

Vedle teorií s širším zaměřením, jako je matematická logika, zabývá se vyšetřováním obecných vlastností zmíněných pojmů již uvedená pravděpodobnostní teorie informace, *teorie jazyků*, *teorie automatů*, *teorie algoritmů*; uvnitř nich se zkoumají speciálnější otázky týkající se *rekurzivních* (tj. *vyčíslitelných*) *funkcí*, *rekurze* vůbec, *sémantiky* a *verifikace* programů, *složitosti* algoritmů, obecných vlastností *procesů*, *automatického dokazování* vět ap. To vše spadá do oblasti teoretické informatiky; její metody mají převážně, byť ne výlučně, matematický charakter; vedle podnětů z konkrétní informatiky

*) Článek vznikl doplněním referátu o informatice, který autor připravil spolu s J. Bečvářem a přednesl dne 28. 11. 1972 na zasedání kolegia ČSAV. Doc. Bečvářovi vděčí autor i za podrobnou recenzi, četné připomínky a řadu formulací.

probíhá i její vnitřní matematický rozvoj, a proto se stala téměř výlučně doménou matematicky erudovaných pracovníků.

Počítačová, resp. konkrétní informatika zahrnuje naproti tomu svým charakterem značně rozmanitou problematiku, jejíž zásadní řešení ve většině případů vyžaduje, resp. by vyžadovalo použití exaktních metod (zvláště matematických), zároveň však často není možné bez soustavně vypěstovaného „citu" pro řešení konkrétních problémů. Využívá ovšem bohatě výsledků teoretické informatiky, využívá však v rostoucí míře i dalších klasických nebo nově vznikajících disciplín, které do vlastní teoretické informatiky obvykle nezařazujeme, jako je *operační analýza, statistika, teorie masové obsluhy, teorie systémů, modelování* a *simulace*, některé partie numerické matematiky apod. (Ostatně i teoretická informatika sama se často obrací k řadě z nich a kromě toho si vypůjčuje např. metody moderní algebry, teorie kategorií, topologie, analýzy atd.)

Přesto se zdá, že v rámci počítačové informatiky nejaktuálnější použití matematiky v současné době zatím nespočívá ani tolik v exploataci nejhlubších matematických výsledků jako spíš v aplikaci základních matematických metod: definování pojmů, formulaci problémů a předpokladů, uplatnění obecného pohledu odsouvajícího do pozadí nepodstatné detaily, ve snaze po vytvoření jednotného deduktivního systému. Hromadné uplatnění těchto zásad při tvorbě a popisu počítačových systémů by odstranilo terminologickou a koncepční roztříštěnost a neopodstatněnou závaznost a složitost řady projektů, manuálů a učebnic. Práce s počítačem by přestala zbytečně vykazovat rysy liturgického obřadu – alespoň v případech, kdy jde o úkony a ideje v podstatě triviální. Existující kritika současného stavu věcí, zejména pokud jde o operační systémy, kde je situace nejméně konsolidovaná, a spolu s tím vzrůstající vliv matematicky erudovaných pracovníků slibuje v nejbližší budoucnosti příchod mohutné nápravné vlny. Zatím však skutečnost, že matematizace praktické počítačové informatiky je na postupu, by neměla zastřít fakt, že převážná většina nynějšího technického i programového vybavení počítačů je stále ještě budována především na základě praxe, intuice a metody zkoušek a omylů ve stylu, který Cox vtipně a výstižně označil jako rokoko.

Při velmi hrubém výčtu můžeme zařadit do konkrétní informatiky tato témata (z nichž každé má bohatě rozčleněnou vnitřní náplň): *struktura dat*, jejich *zobrazení* a *konverze; konkrétní programovací jazyky* a jejich *překladače; operační systémy* a režimy práce s počítačem; celková *architektura* počítačového systému; *hodnocení* a nasazení počítačů. Naproti tomu otázky konstrukce a zvláště technologické záležitosti se obvykle vyčleňují stranou a přesouvají do samostatné inženýrské oblasti.

2.

Dnes již poměrně dost známá *teorie automatů* vychází z představy zařízení, které je schopno transformovat vstupní informaci ve výstupní informaci. Je-li informace kódována konečnou abecedou a přichází-li na vstup v diskrétních časových okamžicích, hovoří se o *diskrétních automatech*. Závisí-li výstup pouze na vstupu v témže časovém okamžiku, hovoříme o *automatu bez paměti*. Obecně může výstup v daném okamžiku záviset i na vstupech v předcházejících okamžicích, na „historii". Má-li mít automat

možnost vzít takovou historii v úvahu, musí být obdařen *pamětí*, ať konečnou (jde pak o *konečné automaty*) nebo potenciálně nekonečnou (s možností postupně zvyšovat objem zapamatované informace – odtud někdy název „*rostoucí*" automaty). Automat se ve své činnosti řídí algoritmem, který může být pevně dán nebo uložen ve formě programu v jeho paměti (pak jde o *univerzální automat;* jeho univerzálnost je ovšem vymezena třídou použitelných programů). Automaty je možno vyšetřovat z hlediska jejich činnosti, tj. z hlediska zobrazení, která realizují, nebo z hlediska jejich struktury (metod organizace paměti a řízení); oba aspekty spolu ovšem úzce souvisí.

Mezi obecné problémy toerie automatů spadají otázky skládání automatů ve větší celky, resp. rozkladu daného automatu na automaty jednodušší (v předepsaném smyslu, se zachováním funkční ekvivalentnosti a obvykle za účelem dosažení menší strukturální složitosti). Z konkrétních otázek tohoto typu se během doby vyvinula relativně samostatná *algebraická teorie atomatů.*

Historicky sehrála teorie automatů pravděpodobně svou první významnou roli na půdě *teorie obvodů*, v níž jde o efektivní konstrukci automatů zpracovávajících informaci zakódovanou pomocí booleovských vektorů (vektorů složených z nul a jedniček). V případě *kombinačních obvodů*, které odpovídají automatům bez paměti a jsou nejdůležitější součástí řady zařízení řídící techniky (a částečně i počítačů), jde o rozklad automatu na technicky dostupné složky, jakými např. po dlouhou dobu byly komponenty realizující základní logické funktory (typu AND, OR, NAND ap.). Zavedením elementárních paměťových prvků, např. zpožďovacích, dostáváme složitější *sekvenční obvody*. Výzkum v tomto směru stále pokračuje, soustřeďuje se však nyní spíše na pracoviště technického zaměření. Po matematické stránce se zdá tato tradiční teorie obvodů v jistém smyslu vyčerpána, nové podněty však přinášejí otázky spojené s integrovanými obvody.

Podstatné pro další vývoj bylo, že studium obvodů (včetně jejich modifikací určených pro modelování nervové činnosti, tzv. *neuronových sítí* aj.) vytvořilo odrazový můstek pro další zobecnění. Jeho základem je abstraktní pojem „*stavu*", v němž je zachycena část dosavadní historie systému podstatná pro další jeho vývoj. Stav reprezentuje obecnou, blíže nestrukturovanou paměť systému. Nový stav závisí na původním i na vstupní informaci. Tento snadno formalizovatelný model abstraktního automatu umožňuje jednoduše a přesně specifikovat a analyzovat systémy nejrůznější povahy na různých úrovních detailnosti; v počítačovém systému je možno jako konečné automaty chápat nejen různé jeho technické jednotky (řadiče aj.), ale i jednotky programové. Tak např. v systému kvazisoučasného provádění více uživatelských modulů (tzv. *multiprogramování*) závisí osud každého z modulů i celého systému na tom, v jakém stavu jsou jednotlivé moduly („provádí se"; „je blokován" – tzn. nemůže pokračovat z jistých důvodů v činnosti; „spí" – tj. není zatím v pořadí na zpracování atp.). Jiným příkladem konečného automatu může být program na předběžné zpracování textů, tj. řetězců znaků; většina překladačů, systémů pro organizaci souborů dat aj. vychází z větších jednotek, než je znak (z identifikátorů, čísel ap.). Ukazuje se, že konečná, předem daná paměť postačuje k tomu, aby seskupení znaků do zmíněných větších celků s eventuální jejich další úpravou mohlo být provedeno; úkol je řešitelný konečným automatem.

Teorie automatů však neposkytuje pouze bohatý arzenál metod analýzy a syntézy

jednotlivých typů automatů; ona vymezuje i hranice jejich působnosti a vytváří jejich *hierarchii*. Na první pohled je např. vidět, že konečný automat nebude schopen správně zpracovat aritmetické výrazy se závorkami, pokud nemíníme klást omezení na jejich složitost: nebude již ani umět rozhodnout, zda počet levých i pravých závorek je týž, což je ovšem nutná podmínka pro korektnost výrazu. (Přivedeme-li na vstup dostatečně mnoho levých závorek, dostane se automat nutně do stavu, ve kterém již byl a ztratí přehled o jejich počtu.)

Jsou proto v dalších modelech konečné automaty opatřovány různými paměťmi. Způsob, jakým je v těchto pamětech informace uchována a z nich vybavována (tzv. *přístupová metoda* v terminologii hromadného zpracování dat) se u klasických modelů obvykle liší od skutečných technických paměťových systémů počítačů. Zatímco u posledních převládá adresovatelná paměť (každý informační záznam je opatřen jménem – adresou, pomocí něhož záznam vybavujeme), u teoretických modelů bývá přístupová metoda dána strukturou paměti a akcí automatu. Je to dáno převážně tím, že původní teoretický model rostoucího automatu, dnes již slavný *Turingův stroj*, vznikl v době, kdy VON NEUMAN o své koncepci reálného počítače teprve uvažoval. (Turingův stroj je konečný automat obohacený o potenciálně nekonečnou lineárně uspořádanou paměť; přitom je-li v nějakém taktu činnosti manipulováno s určitým záznamem, je v dalším taktu uvažován pouze týž nebo sousední záznam. Paměť se nazývá „páska", ukládání a vybavování záznamů provádějí „čtecí/psací hlavy".) Přesto jsme dnes svědky jak tvorby řady modelů, které se blíže přimykají reálným předlohám, tak zpětné snahy převést do oblasti technické realizace ty prvky a ideje, které se v teoretických konstrukcích maximálně osvědčily.

Nejtypičtějším příkladem je tu zřejmě idea *zásobníkového automatu*, jehož paměť je organizována na základě principu „kdo dřív přijde, ten později mele" (LIFO – last in, first out): vybavován je vždy záznam, který byl ukládán jako poslední. Je pravděpodobné, že neexistuje složitější předkladač nebo organizační program, který by této ideje v té či jiné formě nevyužil. Přijde vhod, kdykoliv je nutno přerušit práci na nějakém úkolu a dokončit ji, až bude splněn jistý podúkol: všechny informace rozpracovaného úkolu uložíme v zásobníkové paměti, kde je najdeme, jakmile se vrátíme k jeho dokončení. Zásobníkové organizace použijeme proto, že podúkol může být opět přerušen podpodúkolem atd.; v takovémto systému se dílčí úkoly samozřejmě dokončují v opačném pořadí, než v jakém byly zahájeny. Ilustrace uvedené situace najdeme nejen v „příkladech ze života", ale i při vyhodnocování závorkových výrazů, zpracování blokové struktury programovacích jazyků typu ALGOL, obhospodařování přerušování v operačním systému a jinde. Není proto divu, že existují počítače, u nichž jsou některé paměti organizovány popsaným způsobem přímo technicky (např. B 5500); u ostatních je třeba zásobníkový automat pro uvedené účely simulovat programově.

Méně známá, ale přesto zřetelná ovlivnění praktického vývoje teoretickými úvahami by se dala nalézt v oblasti konstrukce *multiprocesorů* (ať již homogenních, tj. sestávajících z řady stejných kopií, nebo nehomogenních, v nichž některé procesory – automaty hrají speciální úlohu); velkou roli v projektech typu RW-400, Illiac IV, CDC aj. sehrály studie o tzv. *iterativních sítích*, *Hollandových automatech* ap. a ovšem teorie paralelního zpracování, k níž se ještě vrátíme.

Na závěr odstavce ještě konstatujme, že teorie abstraktních automatů je dnes značně rozsáhlá disciplína, která zahrnuje kromě studia nejrůznějších variant nahoře uvedených modelů i automaty s obecnou strukturou paměti (jakožto univerzální modely pro algoritmické zpracování informace), samoreprodukující se automaty, lineární, stochastické, asynchronní, nedeterministické automaty, automaty pracující v náhodném prostředí atd. Na bázi teorie automatů byly také řešeny otázky algoritmické rozhodnutelnosti některých speciálních logických teorií.

3.

Teorie *formálních jazyků* vznikla na půdě praktické lingvistiky, brzy se však ze snahy o formální vystižení faktu, že „věta je podmět + přísudek", „podmět je např. ‚otec', ‚matka', ‚syn'..., přísudek je např. ‚píše', ‚čte', takže ‚otec čte', ‚matka píše' ... jsou příklady vět" — tedy z pokusu o formalizaci faktů známých z páté třídy — vyvinul aparát, který si ve své eleganci, obecnosti a hloubce v ničem nezadá s řadou klasických matematických partií. Poněvadž teorie formálních jazyků současně je základem praktické realizace konkrétních překladačů z programovacích jazyků, lze pochopit pokušení prohlásit teorii jazyků a překladačů za hlavní matematickou disciplínu konkrétní informatiky; přispívá k tomu i skutečnost, že právě tento obor je nejlépe zásoben vhodnou monografickou, učebnicovou i časopiseckou literaturou.

Pro klasickou teorii formálních jazyků je základním pojmem pojem *gramatiky* jakožto soustavy *substitučních pravidel* (nad jistou abecedou), umožňujících v daném *řetězci* (tj. posloupnosti symbolů z této abecedy) nahradit jistou jeho část jiným řetězcem a tento proces *odvozování* opakovat. Na rozdíl od známých Markovových algoritmů je přitom proces odvozování nedeterministický, takže z daného výchozího řetězce obdržíme celou řadu jiných — ty pak tvoří jazyk *generovaný* uvažovanou gramatikou z výchozího řetězce. Pravidla i proces odvozování jsou při tom podrobeny různým omezujícím podmínkám; v abecedách většiny praktických gramatik jsou např. vyčleněny tzv. *metaproměnné*, které hrají roli jmen gramatických celků (*gramatických kategorií*) a musí být v závěrečné fázi procesu odvození eliminovány (srov. rozdíl mezi metaproměnnými typu „věta", „podmět", „přísudek" a mezi vlastními prvky jazyka „otec", „píše"- atd.). Typ pravidel a způsoby zacházení s nimi danou gramatiku charakterizují a klasifikují.

Zvláštní pozornost zasluhují gramatiky s pravidly, která zaručují, že v procesu odvozování nedochází ke zmenšování délky odvozeného řetězce; z toho mj. vyplývá možnost přesvědčit se, zda daný řetězec do jazyka generovaného touto gramatikou patří nebo ne. Každé pravidlo těchto tzv. *kontextových gramatik* je charakterizováno tím, že jeho „levá strana" (určující podřetězec, který bude v procesu odvozování nahrazen) má délku nejvýš rovnou délce „pravé strany" (určující podřetězec, který nastoupí na místo nahrazovaného); přitom se předpokládá, že v nahrazované levé straně je aspoň jedna metaproměnná. Je-li ve všech pravidlech levá strana tvořena právě jednou metaproměnnou, takže příslušné pravidlo je možno použít vždy, kdykoliv se tato metaproměnná vůbec — bez ohledu na to, v jakém kontextu ostatních symbolů — vyskytne,

jde o gramatiku *bezkontextovou*. Velmi speciálním případem bezkontextových gramatik jsou gramatiky *regulární*, které povolují náhradu metaproměnné pouze řetězcem obsahujícím nejvýše jednu metaproměnnou, a to ještě jen na jeho konci. Tyto tři klasické typy gramatik tvoří jen příklady z velkého množství ostatních, jež jsou dnes zkoumány – ovšem příklady stále značně významné.

Na běžné jazyky, ať přirozené či umělé (programovací), se můžeme z hlediska formálního (syntaktického) dívat také jako na množiny řetězce symbolů; otázka je, zda mohou být získány nahoře naznačeným formálním procesem odvozování v rámci jisté gramatiky.

Ukazuje se, že pro většinu programovacích jazyků je situace poměrně příznivá: není-li přímo možné definovat jazyk vhodnou gramatikou, bývá takto možno vystihnout alespoň podstatné rysy jazyka. (Gramatika může generovat jistý nadjazyk a nepřípustné řetězce se eliminují pomocí jednoduchých „sémantických" doplňků. ALGOL 60 byl např. původně popsán bezkontextovou gramatikou doplněnou pravidly typu „každý identifikátor kromě návěští a formálních parametrů má být deklarován", „příkaz skoku nemůže vést zvnějšku dovnitř bloku" ap. Zahrnout tato pravidla do syntaxe by gramatiku podstatně zkomplikovalo a porušilo její bezkontextovost.) Přirozené jazyky naproti tomu vykazují většinou takovou složitost a nepravidelnost, že jejich formální popis může být jen částečný.

Formální definice jazyka není samoúčelná; jejím hlavním cílem bývá snaha ulehčit *analýzu* předloženého řetězce. Vzato opět z hlediska syntaktického je analýza („větný rozbor") proces inverzní k procesu odvozování; má za úkol najít gramatické kategorie, které je možno v řetězci objevit. Např. v triviálním ALGOLovském programu

begin real *x, alfa;*
x := *0; alfa* := *x* + *1* **end**

zjistíme mj. jako nejnižší kategorie písmena (např. *x*, *a*, *f*), jim nadřazené identifikátory (např. *alfa*), aritmetické výrazy (*x* + *1*), příkazy (*alfa* := *x* + *1*), deklarace (**real** *x*, *alfa*) a konečně blok reprezentující celý program.

Tento proces analýzy (zde ilustrovaný ve tvaru „zdola nahoru" – od nejjednodušších celků k vyšším) je velmi důležitý při překladu jednoho jazyka do druhého (např. při překladu z ALGOLu do jazyka stroje). Rozpoznání každé gramatické kategorie totiž odpovídá přesně specifikovaná akce (zjištění identifikátoru odpovídá zanesení do tabulky identifikátorů; příkazu – vytvoření části přeloženého programu; deklaraci – vyhrazení jisté části paměti ap.), takže sestrojit překladač znamená sestavit příslušný *syntaktický analyzátor* a přiřadit vhodné akce jednotlivým gramatickým kategoriím. Poněvadž tyto akce de facto vystihují význam příslušných kategorií, mluvíme o *sémantické specifikaci* překladače.

Není tedy divu, že otázkám syntaktické analýzy se věnuje mimořádná pozornost; zejména se hledají typy gramatik, pro které je algoritmus syntaktické analýzy stále ještě dost jednoduchý a které na druhé straně jsou dost bohaté, aby jimi bylo možno generovat prakticky používané jazyky. Také je žádoucí, aby řetězce jazyka generovaného gramatikou bylo možno analyzovat jednoznačně (a nevznikaly tak konstrukce typu „chléb se solí" – chléb s ČÍM nebo chléb se CO?). Podtřídami bezkontextových gra-

matik, které v řadě praktických případů splňují po ev. předchozích úpravách uvedené požadavky, jsou tzv. *precedenční* gramatiky. Zavedením částečného uspořádání mezi symboly, které vymezuje „přednost při odvozování", je naznačeno, které symboly analyzovaného řetězce máme shrnout do (vyšší) gramatické kategorie. Obecnější slibnou třídou jsou tzv. *LR(k)* gramatiky, u nichž je povoleno při postupném prohlížení textu zleva doprava (Left to Right) využít v daném místě i informaci obsaženou v následujících *k* symbolech ap.

Na zařízení provádějící syntaktickou analýzu je možno pohlížet jako na jistý automat (realizovaný ovšem obvykle programově); vyvstává tedy přirozená otázka, jakého typu automat je schopen analyzovat řetězce generované gramatikou daného typu. Ukazuje se, že souvislosti mezi gramatikami a automaty existují, že jsou přirozené a charakteristické. Tak např. regulárním gramatikám odpovídají v tomto smyslu právě konečné automaty, *LR(k)* gramatikám zásobníkové automaty deterministické (s jednoznačně definovanou činností), zatímco obecným bezkontextovým gramatikám bohužel pouze zásobníkové automaty nedeterministické (u nichž je v některých etapách další krok vybírán — odhadem nebo vyzkoušením více eventualit — z jisté množiny možných). Na nejvyšší úrovni korespondují obecné gramatiky s Turingovými stroji.

4.

Pro *teorii algoritmů* se pozadí počítačů stalo inspirujícím impulsem, který rozsáhle obohatil její náplň.

Všimněme si především otázky *správnosti* (korektnosti) algoritmů, resp. *programů*, pomocí nichž jsou algoritmy zachyceny. Každý programátor praktik ví, že neočekávané chyby se mohou projevit i v programech psaných v jazycích s rozvinutou syntaktickou kontrolou na úrovni překladu a s bohatými prostředky pro sledování výpočtu; zárukou správné funkce není ani dlouhodobé úspěšné používání — nebezpečí latentní chyby prakticky neustále hrozí většině složitějších programových celků. To ovšem vede buď k nedokonalé funkci, nebo k nákladným kontrolním a opravným zásahům; obojí je nežádoucí. Současný rozvoj metod důkazů správnosti programů (a ev. důkazů jejich dalších vlastností, jakou je např. ekvivalence dvou programů) je proto v současné době předmětem zvýšeného zájmu.

Jedna z takových metod se opírá o výběr vhodných tvrzení o vztazích mezi proměnnými, které se účastní výpočtového procesu. Podaří-li se programátorovi formulovat a dokázat tvrzení, které popisuje zamýšlený efekt programu (většinou to bude tvrzení o jistém vztahu mezi vstupními a výstupními proměnnými) a podaří-li se mu dokázat, že program skutečně skončí, bude problém řešen. Všechny zmíněné fáze jsou ovšem problematické; aby byl programátor schopen dokázat své tvrzení, je většinou nucen celý program — představme si jej ve tvaru blokového schématu — rozložit na menší celky a najít (odhadnout) pomocná tvrzení pro tyto celky, přičemž dokázané tvrzení pro určitý blok použije jako předpoklad pro důkaz tvrzení pro blok na něj navazující. Odměnou za práci spojenou s takovým — ev. opakovaným rozkladem, se „strukturová-

ním" programu, je možnost snadnější lokalizace chyby, úpravy a zobecnění jednotlivých bloků ap.

V současné době se konají prvé pokusy o důkazy správnosti vybraných netriviálních algoritmů (byla dokázána správnost speciálního algoritmu pro dělení čísel, třídícího algoritmu a dokonce jistého jednoduchého překladače); vše to ovšem zatím jsou spíš ilustrace navržené metody než akce vyvolaná praktickým popudem. Přesto se zdá, že vstupujeme do období, v němž se metodika programování může stát samostatnou vědeckou disciplínou.

V souvislosti se zmíněnou metodou si povšimněme jedné slibné ideje, i když zvlášť zde jde o úvahy zatím veskrze teoretické. Matematická logika dnes nabízí řadu metod hledajících důkazy (nebo vyvrácení) tvrzení zapsaných v určitém formálním systému. Jestliže se podaří formulovat tvrzení popisující zamýšlený efekt programu ve vhodném formálním systému a jestliže se podaří najít (mechanicky) důkaz tohoto tvrzení, je možno z důkazu zrekonstruovat hledaný program. Pokusy v tomto směru jsou činěny a ukáží-li se někdy jako realistické, dostanou programátoři k dispozici skutečně „*problémově orientovaný*" jazyk; do té doby se zdá být často přetřásaný rozdíl mezi těmito jazyky a „*procedurálně orientovanými*" jazyky dost nejasný.

Padlo zde slovo „strukturování programu". V dnešní době se z tohoto celkem nenápadného termínu začíná stávat pojem, který může hluboce ovlivnit způsob tvorby algoritmů a programů. Na rozdíl od přesně specifikovaných, ale těžko realizovatelných postupů, o nichž jsme právě mluvili, jde v tomto případě spíše „jen" o souhrn jistých metodických pokynů — zato pokynů, jejichž účinnost může okamžitě vyzkoušet každý programátor. K základním principům *strukturovaného programování* patří:

a) Programovat bez užití explicitních skokových („**goto**") příkazů, tj. omezit se na jejich řazení sekvenční, na větvení („**if** ... **then** ... **else**") a opakování (**for** ... **do** ...). Ukazuje se, že tyto tři způsoby členění programu do příkazů teoreticky stačí; omezíme-li se na ně v daném případě i prakticky, přispívá to k přehlednosti programů a zpětně i ulehčuje ev. pokusy o jejich formální verifikaci.

b) Programovat metodou „shora-dolů", tj. začít zápisem algoritmu ve fiktivním jazyce s libovolně mohutnými příkazy a tyto příkazy postupně rozvádět až na úroveň konkrétního daného programovacího jazyka. Současně se zjemňováním příkazů realizovat i odpovídající zjemňování struktur dat (reprezentují „*komunikační zóny*" příkazů). Proti dosud běžnějšímu modulárnímu přístupu (který je v podstatě jednou z možností výstavby zdola nahoru) zde odpadá nutnost testovat a ladit vzájemnou součinnost jednotlivých složek, která bývá nejnáročnější.

c) Považovat každý zvolený jazyk za jazyk jistého *virtuálního počítače* a teprve potom se starat o strukturu tohoto počítače. Nebo opačně: při tvorbě složitých systémů vytvářet postupně hierarchii virtuálních systémů: každý další se opírá o strukturu pouze bezprostředně předcházejícího a je základem následujícího. Tento „*princip abstrakce*" ovšem těsně navazuje na zásadu zmíněnou sub b.

Ukazuje se, že dodržování těchto principů spolu s dokonalou organizační přípravou projektu může podstatně urychlit a zvýšit spolehlivost tvorby velkých programových celků.

K analýze podstaty programů jakožto zápisů algoritmů v posledních letech významně přispívá *teorie programových schémat.*

Programové schéma (též: abstraktní program) je, zhruba řečeno, program, ve kterém všechny příkazy a testy se nahrazují symboly, které označují blíže nespecifikované funkce a predikáty. Při vyšetřování programových schémat si tedy všímáme těch vlastností, které souvisejí pouze se strukturou výpočetního procesu. Každé tvrzení o jistém programovém schématu je současně tvrzením o celé řadě programů, které vzniknou jeho *interpretací.*

Teorie programových schémat je poměrně rozvinutá; většina jejích tvrzení se zabývá otázkami ekvivalence a příp. zjednodušování schémat, otázkami zastavení programů vzniklých interpretací schémat a otázkami rozhodnutelnosti těchto problémů. Jako u mnoha obecných teorií je bohužel i zde většina závažnějších výsledků negativních.

Aktuální části teorie algoritmů je dále ta, která se týká jejich složitosti.*)

Je samozřejmé, že máme-li si vybrat ze dvou algoritmů řešících týž problém, vybereme ten jednodušší. Toto tvrzení zní tak přesvědčivě, že se o něm nechce pochybovat — a přece není tak evidentní. Potíž je v tom, že kritérií složitosti, resp. jednoduchosti může být několik a mohou (a obvykle dokonce budou) si odporovat. Známý je příklad distribuce mezi pamětí a časem (vyhradíme-li větší prostor v paměti, můžeme často docílit zrychlení výpočtu a naopak) či mezi dobou překladu a dobou výpočtu (složitější překladač produkuje efektivnější programy).

Otázky kritérií složitosti výpočtu a algoritmů a jejich precizace v rámci zvolených modelů, ať již konkrétních (automatů, jazyků, schémat) či abstraktních (axiomatických systémů budovaných na půdě teorie rekurzivních funkcí ap.), se intenzívně studují a začínají nést i první prakticky použitelné výsledky.

Dnes již klasické teorémy týkající se algoritmické nerozhodnutelnosti a obdobných pojmů se tak stávají přirozenou součástí systému tvrzení o nemožnosti realizovat určité úkoly na určitých modelech za různých podmínek omezujících kvantitativně jejich možnosti (např. daným limitem na paměť a čas u modelů, u nichž jsou tyto pojmy definovány).

5.

Algoritmy, jejichž vyšetřováním se matematika zabývá od nepaměti, měly ve své převážné většině *sekvenční* charakter — jednotlivé kroky uvažovaného algoritmu následují jeden po druhém, zahájení dalšího je vázáno na ukončení předchozího. Omezit se na studium takovýchto algoritmů bylo po dlouhou dobu zcela přirozené — technická zařízení, která byla k dispozici, byla *monoprocesorová;* počítače měly pouze jednu řídící jednotku a ta byla v daném okamžiku schopna realizovat pouze jeden z příkazů programu. Také mozek (obyčejného) člověka funguje patrně do značné míry sekvenčně — myslet na více věcí naráz vede obvykle spíš k chybě než k úspoře času.

*) Část článku zabývající se teorií programových schémat, složitostí algoritmů, procesy a jejich paralelním během musela být z prostorových důvodů proti původní verzi podstatně zkrácena.

První náznaky možností spojených s překrýváním více operací vznikly ve sféře technické konstrukce. Již realizace jedné strojové instrukce (chápané programátory často jako dále nedělitelná atomická jednotka činnosti) vykazuje při klasickém řešení špatné využití drahého elektronického zařízení: postupně se aktivizují obvody mikroinstrukcí, vybavení instrukce z paměti, dešifrování instrukce, vybavení operandu z paměti, vlastních funkčních jednotek a konečně obvody uložení výsledku do paměti, přičemž vždy zbývající obvody jsou nečinné. Prvním krokem k paralelnímu zpracování bylo tedy osamostatnění modulů realizujících uvedené procesy a tím i možnost překrývaného provádění instrukcí: k zahájení realizace další není třeba čekat na úplné dokončení předchozí. Ještě větší možnosti poskytují systémy, ve kterých některé z modulů-procesorů jsou k dispozici v několika kopiích, takže může dojít k „čistému" paralelnímu běhu dvou procesů. Není ovšem vždy nejjednodušší využít možnosti takto nabízené. Má-li být takový *multiprocesorový systém* dobře využit, je třeba algortimy pro řešení konkrétních problémů upravit pro paralelní způsob zpracování. Zatímco úprava na úrovni mikroinstrukcí je svěřena většinou logice vlastního procesoru, vyšší formy paralelismu připravuje buď překladač, nebo sám programátor.

Nejvíce rozpracované jsou zatím metody pro konverzi aritmetických výrazů do tvaru umožňujícího vysoký stupeň paralelního zpracování.

Mnohem složitější aparát je zapotřebí k odhalení možnosti paralelního provádění několika příkazů. Je většinou orientován na přiřazovací příkazy vyšších programovacích jazyků. Proměnné těchto příkazů se klasifikují podle toho, zda jsou jejich hodnoty pouze čteny, anebo zda jsou uvažovanými příkazy také měněny. Pro jednoduché případy (např. $x := a, y := b$) zajistí možnost paralelní realizace evidentní požadavek disjunktnosti příslušných tříd proměnných. Ve složitějších případech (eliminace nevhodných pomocných proměnných, indexované proměnné, cykly apod.) se však kritéria a jim odpovídající konstrukce značně komplikují.

Na možnosti paralelního zpracování myslela i řada tvůrců programovacích jazyků a umožnila tak samotnému programátoru navrhnout k paralelnímu provedení jisté části algoritmu. Dvojice příkazů FORK – JOIN („rozděl" a „spoj" probíhající proces(y)) byla jedna z prvních. Obdobné možnosti má dnes uživatel PL/1, ALGOLu 68 (kde čárka na místě středníku stačí k naznačení možnosti paralelního zpracování jí oddělených celků) ap. Existují i jazyky (COMPEL – Compute Parallel), které nerozšiřují dosavadní koncepce jazyků o možnosti paralelismu, ale vycházejí z něho přímo jako ze základu; práce s nimi je zatím bohužel dost obtížná.

Celkovému vývoji odpovídají i tendence směřující k navrhování nových nebo k úpravě známých algoritmů z řady konkrétních odvětví (zejména lineární algebry a parciálních difereniálních rovnic) do tvaru vhodného pro paralelní výpočet.

Celá problematika je velmi složitá a náročná; vznikají proto četné teoretické analýzy a modely. Vstup do historie mají např. zaručeny tzv. *Petriho sítě*, *paralelní programová schémata* KARPA - MILLERA ap.

Vzhledem k tomu, že v této oblasti technika, zdá se, předběhla programovou a teoretickou připravenost a vzhledem k celkové koncepční důležitosti tématu se dá očekávat rostoucí zájem matematiků o celou problematiku.

Zmínili jsme se o probíhajících procesech. Termínu „*proces*" se běžně používá k ozna-

čení dynamického protějšku statického zápisu algoritmu v nějakém systému. Uvažme nejčastější případ, totiž procesoru řízeného programem a pracujícího v jisté paměti. Proces lze pak poněkud přesněji definovat jako posloupnost stavů paměti vzniklou *interpretací* programu procesorem, resp. jako systém v čase probíhajících akcí, které tuto posloupnost vytvářejí.

Procesy a jejich vlastnosti jsou často popisovány antropomorfizujícími termíny, které ilustrují jejich dynamičnost. Proces může být „živý" (je prováděn), „spát" (dočasně odložen), „mrtvý" (zrušen), může mít řadu „potomků" (procesů, které vytvoří a ev. oživí), kteří „dědí" některé jeho vlastnosti a možnosti. Má-li např. rodičovský proces k dispozici jisté *zdroje* (procesory, paměti apod.), může věnovat svým „potomkům" některé z nich, nemůže však pro ně požadovat od systému žádné další.

Je až s podivem, jak užitečné se ukázalo být zavedení tohoto pojmu, a to i na celkem nepřesné intuitivní úrovni. Matematizace dynamického popisu práce počítačového systému a jeho subsystémů může v blízké budoucnosti vést ke vzniku samostatné a plodné teorie procesů.

Jsou známy operace, které jsou schopny koordinovat složité systémy na sebe různě navazujících procesů; jde např. o Dijkstrovy tzv. *semafory* a jejich různá zobecnění. Řešit otázku kooperace procesů není jednoduché, mají-li být splněny některé přirozené předpoklady. Jeden z nich požaduje, aby se systém procesů nedostal do slepé uličky tím, že každý z procesů čeká na dokončení některého jiného a soustava tak ustrne na mrtvém bodě. Takováto situace může nastat na křižovatce bez vyznačení přednosti v jízdě, přijedou-li k ní auta současně ze všech čtyř směrů: podle pravidel dává každý řidič přednost sousedu zprava a všichni tedy stojí (pokud se nedohodnou nebo některý neporuší pravidla), ačkoliv je křižovatka volná. Podobné případy mohou nastat i v počítačovém systému, požaduje-li více procesů tytéž zdroje. Metody prevence, detekce a odstraňování takových situací jsou intenzívně studovány a řešeny metodami teorie grafů, operační analýzy ap.

6. Literatura.

Teoretická informatika se dnes již rozrostla do takových rozměrů, že je prakticky vyloučeno podat seznam byť jen nejzákladnějších prací; obtížné je dokonce uvést i seznam seznamů, které jsou různými autory vytvářeny pro specializovanější tematické okruhy. Proto aspoň několik šířeji a hlouběji zaměřených monografií a přehledů:

O široké škále problémů teorie abstraktních automatů pojednává
M. A. Arbib: *Theories of abstract automata*, Prentice Hall 1969.
Praktičtější otázky z oblasti automatů jsou sledovány v knize
U. M. Gluškov: *Úvod do kybernetiky*, Academia Praha 1968.
Vztahy automatů a jazyků popisuje
J. E. Hopcroft, J. D. Ullman: *Formal Languages and Their Relations to Automata*, Addison Wesley 1969,
zatímco až na úroveň praktických aplikací gramatik je dovedena kniha
P. Gries: *Compiler Construction for Digital Computers*, Wiley 1971.
Základní partie matem. logiky a teorie algoritmů najde čtenář vhodně zpracovány v knize

H. HERMES: *Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit*, Springer Verlag 1961
a částečně i v knize
E. MENDELSON: *Introduction to Mathematical Logic*, van Nostrand 1964 (též v ruském překladu).
Na hranici mezi teoretickou a počítačovou informatikou se pohybuje kniha
P. WEGNER: *Programming Languages, Information Structures and Machine Organization*, Mc Graw Hill 1968,
která si všímá vedle úvodu do struktury počítačového systému a jeho programového vybavení i otázek sémantiky ap.

V článku jsme se snažili popsat několik aspoň částečně uzavřených tematických okruhů; proto se nedostalo na velmi důležité konkrétní „technologické" otázky, které určitě spadají do teoretické informatiky metodicky (struktury zobrazení informace, otázky třídění a hledání, tzv. seminumerické algoritmy aj.). Je však nemožné opominout v tomto seznamu ojedinělé dílo, ve kterém se erudovaný matematik zdařile pokusil o přesnou a hlubokou analýzu řady významných pojmů:

D. E. KNUTH: *The Art of Computer Programming, vol. 1—7* (zatím vyšly svazky 1 až 3), Addison Wesley.

O otázkách verifikace a složitosti algoritmů a problematice programových schémat konkrétněji pojednává a bohatou bibliografii uvádí
J. GRUSKA: *Zložitosť, zrozumitelnosť a dokazovanie korektnosti programov a algoritmov*, sborník *Algoritmy vo výpočtovej technike*, 1972;
—: *Analýza algoritmov* Informačné systémy 3, *3* (1974), 207—217.
Problematiku paralelismu shrnuje obdobně článek
J. L.BAER: *A Survey of Some Theoretical Aspects of Multiproccessing*, Computing Surveys vol. 5, 1, (1973), 31—80.
Ideje strukturovaného programování jsou (možná až příliš) zevrubně vyloženy ve sborníku
O. J. DAHL, E. W. DIJKSTRA, C. A. R. HOARE: *Structured Programming*, Academic Press 1972.
S řadou myšlenek o kooperaci procesů i podnětných koncepcí z oblasti „praktické" teoretické informatiky se čtenář setká v několika článcích sborníku
G. A. R. HOARE, R. H. PERROTT: *Operating Systems Techniques*, Academic Press 1972.

A nakonec upozornění pro ty, kteří hledají stručně a přehledně psanou knížku o operačních systémech s trochou nadhledu:

A. J. T. COLIN: *Introduction to Operating Systems*, McDonald/Am. Elsevier 1971.

Slavný francouzský matematik J. L. LAGRANGE (1736—1813) žil až do r. 1787 mimo Francii a nebyl ani francouzským občanem; v porevoluční Francii mohl zůstat jen na základě zvláštního povolení, byl pak velmi aktivním činitelem při organizaci školství. Pokusil se též o důkaz pátého Euklidova postulátu, dokonce četl již svou práci před shromážděnými členy Akademie věd, pojednou se však zarazil a se slovy „Musím o tom ještě přemýšlet" uschoval rukopis a odešel; více už se k tématu nevrátil. Svými pracemi z matematické analýzy a analytické mechaniky si ovšem již dříve získal velkou autoritu, takže si mohl takový odchod ze sálu dovolit.

„Vždy jsem měl rád matematiku, ale v jisté době jsem se hrozně špatně učil. Vždyť jsem přijímací zkoušku na univerzitu složil jen díky tomu, že jsem si bezprostředně před zkouškou připomněl Newtonův binom, ale ničemu jsem nerozuměl." Tak se vyjádřil L. N. TOLSTOJ, který úvahu o filozofických základech matematiky vložil i do svého románu *Vojna a mír*. Ve svých 63 letech si dal vyložit základy diferenciálního a integrálního počtu, couvl však prý před integrací goniometrických funkcí. L. N. Tolstoj studoval na kazaňské univerzitě v době, kdy jejím rektorem byl N. I. Lobačevskij.