Evžen Kindler

Exaktní matematické metody v programovací technice

Pokroky matematiky, fyziky a astronomie, Vol. 17 (1972), No. 6, 307--315

Persistent URL: http://dml.cz/dmlcz/139529

EXAKTNÍ MATEMATICKÉ METODY V PROGRAMOVACÍ TECHNICE*)

EVŽEN KINDLER, Praha

Dnes, po téměř dvacetiletém odstupu, můžeme napsat, že první ročníky matematické olympiády zpopularizovaly do velké míry mezi gymnasisty exaktní matematické metody, to jest přesné dedukce přesně formulovaných vět, vycházející ze stejně přesně formulovaných definic a axiómů. I když byly myšlenky týchž matematických exaktních metod vneseny v téže době do učebních osnov vyšších tříd všeobecně vzdělávacích i odborných škol, nezbylo často vyučujícím než slevit z požadavků přesného formulování definic, vět a důkazů a zůstat u tradičních požadavků na zvládnutí výpočtových a konstruktivních dovedností, a to prostě kvůli naprosté nepřipravenosti studentů na přesné myšlení a často i kvůli nedostatku času pro výklad všeho, to jest jak výpočtových a konstrukčních postupů, tak jejich přesných formulací i důkazů v oné vrcholné matematické formě. Zatím co se na onu formu dívali nadanější žáci jako na zajímavou věc a ostatní žáci jako na nutné zlo, řešitelům matematické olympiády byl stále více jasný význam oné formy pro matematiku jako vědu.

Avšak příchodem samočinných počítačů, což bylo několik let potom, se profese matematiků zaměřila úplně jiným směrem: výzkumným cílem mnohých z nich nebylo již objevit a dokázat nové věty v dané matematické teorii, nýbrž sestavit program či programovací systém pro samočinný počítač. Tedy splněním výzkumného úkolu nebyl již čin pouze intelektuální, ale do jisté míry i materiální. Když šlo o to, jak sestavit program aplikující nějakou numerickou metodu, práce se dělila na dvě fáze: předně se abstraktně formulovala ona numerická metoda v klasických matematických definicích, formulovaly se její vlastnosti v klasických matematických větách a dokázaly v klasických matematických důkazech; druhá fáze spočívala v řemeslném naprogramování metody. První fáze často odpadla, neboť se běžně programovaly numerické metody, které byly abstraktně odvozeny a formulovány již dříve; jindy se odvozovaly v klasické, abstraktní matematice metody nové, ale bez ohledu na to, zda a jak jsou vhodné pro použití na moderní výpočtové technice. Bylo a je však i mnoho programů, které neaplikují žádnou numerickou metodu. Na vysvětlenou uveďme, že to jsou programy zaměřené jinam než na zpracování obvyklých informací zobrazených reálnými čísly. Jsou to například programy pro strojový překlad nebo pro automatickou generaci jiných programů. Takové programy vlastně žádné exaktně formulované matematické věty neaplikovaly; pokud totiž takové věty vůbec byly, říkaly k cíli programu jen velmi málo (např. že výsledek existuje, ale neříkaly, jak k němu dojít).

*) Tento článek byl původně napsán jako vzpomínkový příspěvek do sborníku vydaného k 20. výročí matematických olympiád v Československu. Později však bylo rozhodnuto, že pro větší rozsah i větší náročnost se hodí spíše pro čtenáře Pokroků matematiky, fyziky a astronomie, kterým jej nyní předkládáme.

Zdálo se, a ještě i nyní se zdá, často oprávněně, že vývoj v tvorbě programů a programovacích metod jde příliš rychle na to, aby mu stačila klasická, předpočítačová matematická věda se svými racionálními, ale nyní jakoby těžkopádnými metodami.

Po promoci jsem pracoval na konstrukci a programovém vybavení nových počítačů. A jako účastníka prvních dvou ročníků matematické olympiády mě přirozeně hned od počátku pálil onen protiklad mezi exaktní metodou matematiky, která mi při matematické olympiádě tak učarovala a jejíž použití jsem si upevňoval během celého vysokoškolského studia na Karlově universitě, a mezi požadavky svého zaměstnání. Výsledkem toho byla aplikace matematických metod v automatickém programování, o němž řekneme dále něco bližšího.

1. Automatické programování

Programem pro samočinný počítač rozumíme napsanou posloupnost instrukcí takového počítače. Toto vymezení není definicí ve smyslu klasické matematiky, ale stačí úplně těm, kteří umějí se samočinným počítačem zacházet a nechtějí to obecně popisovat v exaktních vědách. Pro každý typ samočinného počítače je program obecně něco jiného, neboť dva různé typy samočinných počítačů mají obvykle i různé instrukce. Protože sestavení programu je věc dosti namáhavá, používá se techniky tak zvaného automatického programování. Tato technika je popsána například v závěrečné kapitole knihy [1]; zde si připomeňme jen jednu z jejích částí: Máme-li vyčíslit na samočinném počítači hodnotu nějakého algebraického výrazu, za jehož proměnné jsou dosazena jistá reálná čísla, můžeme to udělat na samočinném počítači pomocí programu, který odpovídá danému algebraickému výrazu a dá výsledek v závislosti na datech, to jest na reálných číslech, která mají být dosazena za proměnné onoho algebraického výrazu; odpovídající program může být sestaven ručně, je však výhodné sestavit jednou pro vždy jediný program, který pracuje takto: přečte si na příklad z děrných štítků text odpovídající nějakému algebraickému výrazu, uloží si jej do paměti, a pak do jiných míst paměti sestaví odpovídající program, který vyčísluje onen algebraický výraz na tomtéž počítači*). Jde tedy vlastně o strojový překlad z jazyka tradiční algebry do jazyka daného samočinného počítače. Postup, který je v onom jediném, tak zvaném generujícím programu neboli kompilátoru naprogramován, se nazývá překládacím algoritmem. A když je jednou takový algoritmus vymyšlen, může být oprávněn požadavek důkazu, že onen algoritmus je bez chyby.

A tento požadavek nabízí možnost aplikovat metody klasické matematiky. K tomu je tedy nutno přesně definovat, co to je algebraický výraz (jaká pravidla řazení symbolů se tedy musí respektovat, aby šlo o algebraický výraz a ne třeba o cito-

*) Taková činnost počítače má pak dvě fáze: v prvé fázi počítač sestaví program, v druhé jej provede.

slovce – tzv. syntaxe algebraického výrazu) a jeho hodnota (při dosazení za jeho proměnné – tzv. sémantika algebraického výrazu), dále je třeba definovat přesně, co to je program (tedy opět jeho syntaxi a sémantiku) a formulovat v přesných a přesně dokázaných větách, že daný překládací algoritmus sestaví na základě každého algebraického výrazu po konečně mnoha krocích program, jehož výsledkem je – a to pro libovolné dosazení za proměnné – odpovídající hodnota původního algebraického výrazu.

Potřebné věty, definice a důkazy jsou publikovány v práci [2]. Zde jen uveďme v krátkosti definice a věty a vysvětleme jejich funkci v celkové koncepci důkazu.

2. Syntaxe

2.1. Nejprve definujeme syntaxi algebraického výrazu; provedeme to indukcí:

2.1.1. Posloupnost obsahující jediný abecední znak je činitel; nazýváme ji proměnnou.

2.1.2. Každá posloupnost znaků, která je činitelem, je členem.

2.1.3. Každá posloupnost znaků, která je členem, je algebraickým výrazem.

2.1.4. Nechť v posloupnosti znaků $a_1, a_2, \ldots, a_{n-1}, a_n, a_{n+1}, \ldots, a_m$, kde $1 < n < m$, je $a_1, \ldots, a_{n-1}$ člen, $a_{n+1}, \ldots, a_m$ je činitel a a_n je znak pro násobení nebo pro dělení; pak celá posloupnost $a_1, \ldots, a_m$ je členem.

2.1.5. Nechť v posloupnosti znaků $a_1, \ldots, a_{n-1}, a_n, a_{n+1}, \ldots, a_m$, kde $1 < n < m$, je $a_1, \ldots, a_{n-1}$ algebraický výraz, $a_{n+1}, \ldots, a_m$ je člen a a_n je znak pro sčítání nebo odčítání; pak celá posloupnost $a_1, \ldots, a_m$ je algebraický výraz.

2.1.6. Nechť v posloupnosti znaků $a_1, a_2, \ldots, a_{n-1}, a_n, a_{n+1}, \ldots, a_m$, kde $2 < n < m - 1$, je $a_2, \ldots, a_{n-1}$ člen, $a_{n+1}, \ldots, a_{m-1}$ je činitel a a_n je znak násobení nebo dělení; nebo nechť v téže posloupnosti je $a_2, \ldots, a_{n-1}$ algebraický výraz, $a_{n+1}, \ldots, a_{m-1}$ je člen a a_n je znak sčítání nebo odčítání. Když a_1 je levá závorka a a_m pravá závorka, pak celá posloupnost $a_1, \ldots, a_m$ je činitelem.

Příklady: A je algebraický výraz, neboť podle 2.1.1. je činitelem, a tedy podle 2.1.2. členem, a pak podle 2.1.3. algebraickým výrazem. Podobně B nebo C je algebraickým výrazem a rovněž činitelem a členem. Podle 2.1.4. je A/B členem, ale nedokážeme, že je činitelem. Podle 2.1.6. je však činitelem (a tedy i členem) (A/B). $A/B + C$ je podle 2.1.5. algebraickým výrazem, stejně jako $A + B$ nebo $C + A/B$, $C + (A/B)$: stejnou cestou bychom dokázali, že i $A + B/C$, $A + A$ a podobně jsou algebraické výrazy, nikoli však činitelé a ani členy. Ty z nich podle 2.1.6. uděláme tak, že je uzavřeme do závorek, takže pak mohou tvořit část delšího algebraického výrazu, např. $(A/B + C)/(A + B/C)$. Posloupnost znaků $+ + AC/-A$ není algebraickým výrazem stejně jako třeba $A/-B$.

Dále definujme funkci p na množině všech znaků:

2.2. Pro znak násobení a dělení nabývá funkce p hodnoty 2, pro znak sčítání a odčítání nabývá hodnoty 1, pro pravou závorku hodnoty nula a pro ostatní znaky nabývá hodnoty rovné 3. Tato funkce se nazývá *prioritou* znaku a exaktně připravuje použití pravidla, jehož obvyklá formulace zní, že operace násobení a dělení mají při vyčíslování algebraických výrazů přednost před operacemi sčítání a odčítání. Operačním znakem budeme dále rozumět znak pro sčítání nebo odčítání nebo násobení nebo dělení (poznamenejme, že vynechávání znaku pro násobení, běžné v tisku, se při použití samočinných počítačů většinou nepřipouští, píšeme jej tradičně $\times$).

2.3. Musíme definovat i *program*. K tomu cíli zavedeme tak zvané pomocné proměnné, to jest znaky dejme tomu $T_1, T_2, \ldots$. Ty považujeme za jakési úplně nové znaky, které se však chovají jako proměnné, vyskytující se v původním, zadaném algebraickém výraze, ač samy v něm nejsou. Budeme je do něho vkládat později.

2.4. Program je konečná posloupnost $I_1, \ldots, I_n$, kde I_j jsou tak zvané instrukce, to jest trojice znaků $a_1^{(j)}, a_2^{(j)}, a_3^{(j)}$; přitom požadujeme, aby $a_2^{(j)}$ byl nějaký operační znak a $a_1^{(j)}$ a $a_3^{(j)}$ byly nějaké proměnné nebo pomocné proměnné. V posledním případě, je-li $a_1^{(j)}$ nebo $a_3^{(j)}$ dejme tomu T_i, musí být $i < j$.

Intuitivní význam pojmu program si vysvětlíme na příkladě: je-li I_j tvaru $a + b$, pak to znamená, že součet proměnných se má dosadit za pomocnou proměnnou T_j. Program se provádí tak, že se vyčíslí instrukce za sebou v tom pořadí, jak jsou uspořádány v posloupnosti $I_1, \ldots, I_n$, uvedené v definici programu. Z toho plyne omezení, uvedené v poslední větě odstavce 2.4. Matematicky přesně reflektované pojmy provádění programu a vyčíslení instrukce budou však uvedeny až v následující části, v sémantice. Příklad malého programu:

$$A + B\,, \quad T_1 - C\,, \quad A - D\,, \quad T_2/T_3\,, \quad T_4 + E \tag{2.4.1.}$$

3. Sémantika

Nejprve definujme sémantiku algebraického výrazu; provedeme to stejnými indukčními kroky jako v definici 2.1.

3.1. Nechť f je reálná funkce na množině všech proměnných (intuitivně: *dosazení reálných čísel za proměnné*). Toto zobrazení rozšíříme na všechny algebraické výrazy:

3.1.1. Nechť posloupnost znaků $a_1, \ldots a_m$ je stejné struktury jako v 2.1.4. Pak

$$f(a_1, \ldots, a_m) = f(a_1, \ldots, a_{n-1}) \,.\, f(a_{n+1}, \ldots, a_m)$$

nebo

$$f(a_1, \ldots, a_m) = f(a_1, \ldots, a_{n-1})/f(a_{n+1}, \ldots, a_m)\,,$$

podle toho, zda a_n je znak pro násobení nebo pro dělení.

3.1.2. Nechť posloupnost znaků $a_1, \ldots, a_m$ je stejné struktury jako v 2.1.5. Pak

$$f(a_1, \ldots, a_m) = f(a_1, \ldots, a_{n-1}) + f(a_{n+1}, \ldots, a_m)$$

nebo

$$f(a_1, \ldots, a_m) = f(a_1, \ldots, a_{n-1}) - f(a_{n+1}, \ldots, a_m),$$

podle toho, zda a_n je znak pro sčítání nebo odčítání.

3.1.3. Nechť posloupnost znaků $a_1, \ldots a_m$ je stejné struktury jako v 2.1.6. Pak

$$f(a_1, \ldots, a_m) = f(a_2, \ldots, a_{n-1}) + f(a_{n+1}, \ldots, a_{m-1})$$

nebo

$$f(a_1, \ldots, a_m) = f(a_2, \ldots, a_{n-1}) - f(a_{n+1}, \ldots, a_{m-1})$$

nebo

$$f(a_1, \ldots, a_m) = f(a_2, \ldots, a_{n-1}) \times f(a_{n+1}, \ldots, a_{m-1})$$

nebo

$$f(a_1, \ldots, a_m) = f(a_2, \ldots, a_{n-1}) / f(a_{n+1}, \ldots, a_{m-1})$$

podle toho, zda a_n je znak sčítání, odčítání, násobení nebo dělení.

Upozorňujeme čtenáře, že zde nejde o funkce několika proměnných, nýbrž o funkci jedné proměnné, jíž je však posloupnost znaků. Tyto znaky oddělujeme, jak je zvykem, čárkami. Například, je-li $f(A) = 3$ a $f(B) = 4$, pak $f(A + B) = 7$, $f(A + B/B) = 1{,}75$, $f(A - B + A) = 10$. $f(- -B)$ není ovšem definováno*).

3.2 Poznamenejme, že zatím není dokázáno, že každou reálnou funkci f na množině proměnných lze rozšířit na libovolný algebraický výraz jednoznačně. Nebudeme to ve formulaci překládacího algoritmu potřebovat, stejně jako v prvých krocích důkazu jeho správnosti. Intuitivně může být jednoznačnost rozšíření funkce f z množiny proměnných na množinu algebraických výrazů patrna, neboť definice 3.1. odpovídá výpočtu hodnoty daného algebraického výrazu při dosazení f za jeho proměnné.

Definice sémantiky programu je mnohem jednodušší:

3.3. Nechť f je reálná funkce na množině proměnných, $P = I_1, \ldots, I_n$ je program. Pak funkci f rozšíříme na množinu pomocných proměnných $T_1, \ldots, T_n$ takto (opět jde o definici indukcí): nechť $I_j = a_1^{(j)}, a_2^{(j)}, a_3^{(j)}$ a nechť $f(a_1^{(j)}) = X$ a $f(a_3^{(j)}) = Y$. Pak $f(T_j)$ budiž $X + Y$ nebo $X - Y$ nebo $X \,.\, Y$ nebo X/Y, podle toho, zda $a_2^{(j)}$ je znak pro sčítání, odčítání, násobení nebo dělení. $f(T_n)$ nazýváme *výsledkem programu P při dosazení f za jeho proměnné.* Na rozdíl od sémantiky algebraických výrazů je okamžitě patrné, že výsledek programu je definován vždy jednoznačně.

Příklad: Výsledkem programu (2.4.1.), uvedeného na konci části 2, při dosazení $f(A) = 7, f(B) = 3, f(C) = 5, f(D) = 6, f(E) = 4$ je číslo 9, protože podle definice

*) V těchto i dalších příkladech bychom měli pást důsledněji $f(A, +, B) = 7$, $f(A, +, B, /, B) = 1{,}75$ atd. Pro lepší čitelnost však čárky, které oddělují jednotlivé znaky v argumentu, vynecháváme.

dostáváme postupně:

$$f(T_1) = f(A) + f(B) = 7 + 3 = 10$$
$$f(T_2) = f(T_1) - f(C) = 10 - 5 = 5$$
$$f(T_3) = f(A) - f(D) = 7 - 6 = 1$$
$$f(T_4) = f(T_2)/f(T_3) = 5/1 = 5$$
$$f(T_5) = f(T_4) + f(E) = 5 + 4 = 9$$

Vidíme, že program vlastně vyčísluje hodnoty algebraického výrazu

$$(A + B - C)/(A - D) + E$$

A to už je souvislost, která nás přivádí k hlavnímu tématu této stati, k algoritmu, který jakémukoliv algebraickému výrazu přiřazuje odpovídající program, který dá jako výsledek jeho hodnotu při jakémkoliv dosazení za proměnné, které se v onom algebraickém výrazu (a v příslušném programu) vyskytují.

Poznámka: Při některých dosazeních může vyjít v sémantice algebraického výrazu i odpovídajícího programu jmenovatel rovný nule. Tento případ lze buď vyřešit tím, že taková dosazení nepřipustíme, nebo jej lze postavit na roveň ostatním případům tak, že zavedeme jako výsledek dělení nulou nějaké pevné číslo, např. nulu nebo jedničku. Na tom nezáleží, neboť žádné zákony aritmetiky, s nimiž by se takové zobecnění dostalo do sporu, nepoužíváme.

4. Překládací algoritmus

V této části popíšeme algoritmus, který z libovolného algebraického výrazu sestaví program, jak bylo právě uvedeno výše. Algoritmus se skládá ze dvou cyklů, které jsou zařazeny v sobě tak, že jeden cyklus (nazvěme ho cyklem A) vyvolá v každém kroku obecně několikrát druhý cyklus (nazvěme jej B), a pak sestaví v tomtéž kroku cyklu A jednu instrukci programu. Cyklus B pracuje takto: za n se dosadí 1 a testuje se znak a_n algebraického výrazu $a_1, \ldots, a_m$. Zjistí se, zda $n + 1$ je menší než m, a jestliže ano, zjistí se ještě, zda $p(a_n) < p(a_{n+2})$. Není-li a_n znak operace nebo je-li $n + 2$ menší než m a současně $p(a_n) < p(a_{n+2})$, zvýší se n o jedničku a celý postup se opakuje. V ostatních případech práce cyklu B končí.

Jeden krok cyklu A vypadá takto: Vyvolá se cyklus B, jak je uveden v předešlém odstavci, a když se najde takové n, že a_n je znak operace a buď $p(a_n) \geqq p(a_{n+2})$ nebo $n + 2 > m$, sestaví se další, dejme tomu i-tá instrukce konstruovaného programu, a to trojice symbolů a_{n-1}, a_n, a_{n+1}; nato se znak a_{n-1} nahradí pomocnou proměnnou T_i a znaky a_n a a_{n+1} se vypustí, tedy celý algebraický výraz se o dva znaky zkrátí a indexy znaků a_{n+2} a dalších se zmenší o 2. Je-li po takovéto úpravě a_{n-2} levá závorka a a_n pravá závorka, vypustí se také ještě tyto dva symboly. Je-li $m = 1$,

práce algoritmu končí (jinými slovy – právě byla sestavena poslední instrukce). Je-li $m \geqq 2$, pak cyklus A provede další krok, a to stejným způsobem, jak bylo právě popsáno, ale na posloupnosti o 2 či 4 symboly kratší.

Příklad: Uvažujme algebraický výraz $(E + F \times G)/(E - (G + H))$. První krok cyklu A, aplikovaného na tento výraz, provede několik kroků cyklu B, až najde znaménko $+$. To má však menší prioritu než symbol následující ob jeden symbol dále (znaménko násobení), takže cyklus B pokračuje v dalších krocích, až u toho znaménka násobení zjistí, že má větší prioritu než pravá závorka, která je po něm druhým následujícím symbolem. Sestaví se tedy instrukce $I_1 = F \times G$ a původní algebraický výraz se změní na $(E + T_1)/(E - (G + H))$. Tím končí první krok cyklu A. V druhém kroku cyklu A se opět po několika krocích cyklu B najde znaménko $+$ jako vyhovující (druhý znak za ním má menší prioritu – je to opět pravá závorka), a tak se sestaví další instrukce $I_2 = E + T_1$. Zpracovávaný algebraický výraz se změní na $(T_2)/(E - (G + H))$ a pak hned na $T_2/(E - (G + H))$. V dalším kroku A se během cyklu B najde postupně znaménko dělení, které nevyhovuje, protože proměnná E, následující dále ob jeden symbol, má prioritu větší, pak znaménko odčítání, které nevyhovuje z týchž důvodů, až konečně znaménko sčítání, které je vyhovující. Sestaví se tedy instrukce $I_3 = G + H$ a zpracovávaný výraz se zkrátí postupně na $T_2/(E - (T_3))$ a na $T_2/(E - T_3)$. V dalším kroku cyklu A se pak přejde znaménko dělení podobně jako v kroku předešlém a na základě vyhovujícího znaménka odčítání se utvoří instrukce $I_4 = E - T_3$ a z výrazu dostaneme nejprve $T_2/(T_4)$ a pak T_2/T_4. Další krok cyklu A pak brzy najde znaménko dělení jako vyhovující (jeho pořadové číslo zvětšené o 3 už překročí délku zpracovávaného výrazu $m = 3$), takže se sestaví instrukce $I_5 = T_2/T_4$ a práce algoritmu končí. Je zřejmé, že při libovolném dosazení za proměnné E, F, G, H je hodnota algebraického výrazu $(E + F \times G)/(E - (G + H))$ rovna výsledku programu

$$F \times G\,, \quad E + T_1\,, \quad G + H\,, \quad E - T_4\,, \quad T_2/T_4\,.$$

K tomu, abychom však dokázali, že taková rovnost platí pro jakýkoliv algebraický výraz a program, generovaný z něho popsaným algoritmem, je třeba dokázat tyto 4 věty:

4.1. v cyklu B se vždy najde hledaný znak;

4.2. krok A se opakuje konečněkrát;

4.3. trojice symbolů a_{n-1}, a_n, a_{n+1}, která má tvořit další instrukci, konstruovanou podle popisu cyklu A, splňuje opravdu požadavky na instrukci, vyslovené v 2.4;

4.4. pro každé dosazení za proměnné je hodnota algebraického výrazu rovna výsledku sestaveného programu (z toho plyne okamžitě tvrzení o jednoznačnosti určení hodnoty algebraického výrazu, které zatím nebylo dokázáno).

Bod 4.1. je zřejmě splněn, neboť jistě poslední operační znak v celé posloupnosti znaků, tvořící algebraický výraz, má požadované vlastnosti. Taktéž i bod 4.2. je splněn, neboť při každém kroku cyklu A se délka zpracovávaného výrazu zmenší

nejméně o 2. Je tedy třeba dokázat platnosti bodů 4.3. a 4.4., což provedeme v další části.

Poznámka: Jak je čtenáři jistě patrno, algoritmus je prázdný pro algebraické výrazy o délce rovné 1 (pouhé proměnné). Tam ani tvrzení o správnosti algoritmu nemá význam, neboť prázdný program nemá definován výsledek. V použití samočinných počítačů tento případ nemá význam. Matematicky přesně se celá záležitost vyjádří například tím, že správnost algoritmu se dokáže jen pro tak zvané složené termy, jak bude ukázáno dále.

5. Postup důkazu

Nejprve zavedeme tyto pomocné definice:

5.1. Algebraické výrazy, které nejsou proměnné, nazýváme složenými termy. Množinu složených termů dělíme na dvě disjunktní části: množinu termů 1. třídy, obsahující právě termy tvaru a_1, a_2, a_3, kde a_1 a a_3 jsou proměnné a a_2 je operační znak, a termy tvaru a_1, a_2, a_3, a_4, a_5, kde a_2 a a_4 jsou proměnné, a_3 je operační znak, a_1 je levá závorka a a_5 je pravá závorka, a množinu termů druhé třídy obsahující ostatní složené termy.

Příklad: $A + B$ je term první třídy, (C/D) také, kdežto $A + B - C$ nebo $C/(B + A)/D$ jsou termy druhé třídy.

5.2. Srdce složeného termu je posloupnost znaků definovaná indukcí: srdce termu první třídy je posloupnost znaků totožná s daným termem. Je-li X složený term a Y člen, pak srdce termu druhé třídy $X + Y$ je rovno srdci termu X. Je-li X proměnná a Y složený term, pak srdce termu druhé třídy $X + Y$ je rovno srdci termu Y. Podobně pro další možnosti indukčního kroku popsaného v bodech 2.1.4. a 2.1.6.

Příklad: Srdcem termu $A + B$ je $A + B$, srdcem termu (C/D) je (C/D), srdcem termu $A + B - C$ je $A + B$, srdcem termu $C/(B + A)/D$ je $(B + A)$. Platí tyto věty (dokázaly by se indukcí stejnými kroky jako definice srdce):

5.3. Člen, který není proměnnou, začíná buď levou závorkou, nebo začíná proměnnou, za níž následuje hned znak pro násobení nebo dělení.

5.4. Nechť $a_1, \ldots, a_n$ je složený term. Jeho srdce je buď trojice znaků a_i, a_{i+1}, a_{i+2}, nebo pětice znaků $a_{i-1}, a_i, a_{i+1}, a_{i+2}, a_{i+3}$ s těmito vlastnostmi: i je nejmenší přirozené číslo, pro které a_i a a_{i+2} jsou proměnné, a_{i+1} je znak operace a buď $i + 3 > n$, nebo $p(a_{i+1}) \geqq p(a_{i+3})$. Je-li a_{i-1} levá závorka a a_{i+3} pravá závorka, je srdce uvedená pětice, jinak je to uvedená trojice. Číslo i nazýváme centrálním indexem složeného termu $a_1, \ldots, a_n$.

5.5. Nechť X je složený term $a_1, \ldots, a_n$, i jeho centrální index. Pak posloupnost Y, vzniklá tak, že a_i nahradíme nějakou pomocnou proměnnou T_j a ostatní znaky srdce vypustíme, je algebraickým výrazem. Je-li původní term činitelem (resp. členem), je i Y činitelem (resp. členem). Y nazýváme j-tou redukcí termu X.

5.6. Nechť X je složený term jako v 5.5. Nechť i je nejmenší index, pro nějž platí: a_i je znak operace a buď $i + 3 > n$, nebo $p(a_{i+3}) \leqq p(a_{i+1})$. Pak $1 < i < n$, a_{i-1} a a_{i+1} jsou proměnné a $i - 1$ je centrální index termu X.

5.7. Nechť X je složený term s centrálním indexem i, Y jeho k-tá redukce a f libovolné dosazení za proměnné, které se vyskytují v X. Nechť však mezi nimi není T_k. Pak lze rozšířit dosazení i za tuto proměnnou předpisem $f(T_k) = F(f(a_i), f(a_{i+2}))$, kde funkce F značí součet, rozdíl, součin či podíl obou argumentů, podle toho, jakou operaci značí znak a_{i+1}. Hodnota algebraického výrazu Y při takto rozšířeném dosazení se rovná hodnotě termu X při původním dosazení.

5.8. Nechť X je složený term jako v 5.5. Definujme posloupnosti $I_1, \ldots I_p$ a $X_1, \ldots, X_p$ takto: $X_1 = X$; je-li X_j složený term $b_1, \ldots, b_m$ s centrálním indexem i, pak existuje X_{j+1} a je rovno $(1 + j)$-té redukci termu X_j, I_j je uspořádaná trojice b_i, b_{i+1}, b_{i+2}. Platí předně, že $I_1, \ldots, I_p$ je program; dále platí: nechť f je libovolné dosazení za proměnné termu X_j, rozšířené na T_{j+1} jako v 5.7. Po konečně mnoha krocích je lze rozšířit na všechna X_j a platí $f(X) = f(X_p)$. X_p je jediný znak, a to T_p. Tedy výsledek programu $I_1, \ldots, I_p$ je roven hodnotě termu X.

6. Závěry

Program $I_1, \ldots, I_p$ z 5.8. lze – podle 5.6. – konstruovat také postupem popsaným v části 4 jako algoritmus (postup udaný v 5.8. totiž algoritmizovaný není). Tím je platnost bodů 4.3. a 4.4. dokázána, a tedy je dokázána i správnost algoritmu.

Příklad na tvoření algebraických výrazů a jejich překládání do programů, který zde byl právě uveden, je záměrně omezen na výrazy aritmetické. Čtenář, který se setkal s jinými algebraickými operacemi (např. booleovskými), si celou látku snadno může interpretovat i pro takové operace. Ba dokonce můžeme překládat výrazy, které jsou smíšeny ze složek aritmetických, booelovských a relačních. Relační složky (např. „větší než", „rovno", „nerovno") tvoří přechod mezi složkami booleovskými a aritmetickými: operuje se na aritmetických složkách a výsledek může tvořit složku booleovskou. I takové výrazy lze tvořit a překládat, musíme jen vhodně zavést pojem priority. To už však překračuje rámec tohoto článku, více se o tom může čtenář dozvědět např. v [3].

Literatura

[1] J. Backus a kolektiv: *Programování v jazyku ALGOL*. Praha, SNTL, 1963, str. 105—120.

[2] E. Kindler: Simple algorithm for programming of arithmetic expressions. Stroje na zpracování informací, sborník *8*, NČSAV, Praha 1962, str. 143—154.

[3] E. Kindler: Translation of arithmetic expressions by EPOS ALGOL compiler. Stroje na zpracování informací, sborník *9*, NČSAV, Praha 1963, str. 79—90.