13. DOKAZOVÁNÍ PROGRAMU

Jedním z důležitých cílů současných trendů v algoritmizaci a programování je snaha o dokazování správnosti programu. Programování lze považovat za nové odvětví aplikované matematiky a program za rozsáhlý teorém. Výzkumem v této oblasti se již delší dobu významně zabývá Dijkstra a jeho dosavadní výsledky, jež v mnohém směru vnášejí zcela nový pohled na algoritmizaci, programování i programovací jazyky, jsou uvedeny v jeho publikaci [1]. Tato kapitola si klade za cíl shrnout nejzávažnější poznatky v dokazování programů a ukázat jejich aplikaci na několika příkladech tak, jak byly uvedeny v [2].

13.1. Programovací jazyk

Zobecnění výpočetního úkonu lze chápat dvojím způsobem. Vezměme si jako příklad mechanismus pro výpočet největšího společného dělitele čísel 111 a 259. Mechanismus operující s čísly 111 a 259 lze zobecnit dvěma způsoby :

a) rozšířit a explicitně stanovit třídu úloh operujících nad stejnými argumenty (rozšířit daný mechanismus o nejmenší společný násobek čísel 111 a 259, jejich součin, součet atd.)

b) rozšířit třídu argumentů, pro niž bude platný daný výpočetní úkon.

Pro účely dokazování správnosti výpočetního úkonu je nesporně vhodnější druhý způsob abstrakce. Mechanismus, který by produkoval výsledky nejrůznějších funkcí hodnot 111 a 259, by se s každým rozšířením třídy úloh dokazoval obtížněji. Podobnou vlastnost nemá rozšíření třídy argumentů.

Vzhledem k tomu, že cílem je správnost algoritmu formou důkazu, není vhodná jeho definice slovní formou. Pro definici algoritmu se hledá vhodná formální notace. Její nejdůležitější vlastností je skutečnost, že dovoluje pracovat s algoritmy jako s matematickými objekty. Umožňuje např. dokazovat teorémy o třídách algoritmů, protože jejich popis má určitou shodnou strukturální vlastnost. Algoritmům zapsaným za účelem zpracování na počítači se říká program a formální notaci pro jejich definici se již v padesátých letech začalo říkat "programovací jazyk". Spojení notace algoritmů s pojmem "jazyk" mělo své přednosti, ale i závažné nevýhody. Na jedné straně byla v té době jazykověda velmi rozvinutým vědním oborem se svou terminologií i metodologií. Na druhé straně "přirozené" - neformalizované jazyky, jimiž se zabývala, čerpají svou mocnost i nedostatky právě ze své neurčitosti a nepřesnosti! Z historického hlediska byla skutečnost, že programovacího jazyka může být použito jako prostředku pro řízení existujících počítačů, považována za jejich nejdůležitější vlastnost. Hlavním kriteriem kvality jazyka byla účinnost, s níž byly programy zapsané v tomto jazyce řešeny na počítači. Důsledkem této skutečnosti je nezřídka se vyskytující odraz nejrůznějších anomálií a technických zvláštností existujících počítačů v programovacích jazycích. Tento vliv způsobuje zbytečnou další intelektuální námahu při tvorbě programů v takových jazycích. Nový přístup k programovacím jazykům se snaží obnovit rovnováhu v tomto smyslu: skutečnost, že algoritmus může být řešen počítačem se považuje za užitečnou okolnost, která nezaujímá nezbytně centrální postavení ve všech úvahách. Programovací jazyk je především prostředek popisu potenciálně velmi složitého abstraktního mechanismu. Nejvýznamnější vlastností algoritmu je kompaktnost jeho argumentů. Na ní závisí důvěra v obecnost výpočetního mechanismu a tedy i spolehlivost vytvořeného programu. Jakmile se poruší nebo ztratí tato kompaktnost, ztrácí algoritmus právo na existenci. Udržení této kompaktnosti je tedy prvořadým úkolem, který musí sledovat i volba programovacího jazyka.

13.2. Základní matematický aparát

Předpokládejme, že v průběhu výpočtu největšího společného dělitele dvou přirozených čísel X,Y projdeme stavy x,y, pro něž platí :

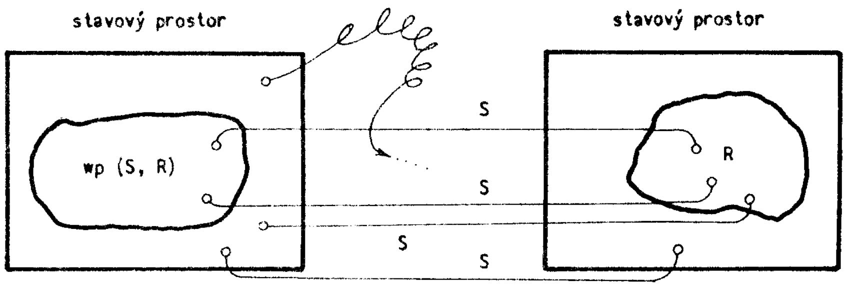
$$NSD(x,y) = NSD(X,Y) \wedge \emptyset < x \leq X \wedge \emptyset < y \leq Y$$

kde NSD je označení funkce největšího společného dělitele, X,Y jsou konstanty pro určitý výpočet a určují počáteční hodnoty proměnných x,y. Podobným vztahům budeme říkat "podmínky" nebo "predikáty".

Jestliže se systém po skončení své aktivity určitě dostane do stavu splňujícího podmínku P, pak říkáme, že systém určitě ustaví pravdivost P. Každý predikát je definován v každém bodu stavového prostoru za předpokladu, že v každém bodu tohoto prostoru má hodnotu "true" nebo "false". Nadále budeme predikáty používat pro označení množiny takových bodů stavového prostoru, v nichž je predikát pravdivý.

O predikátech P a Q říkáme, že jsou si rovny ("P=Q"), jestliže označují stejnou podmínku nebo jestliže označují stejnou množinu stavů. Dále budeme používat dva speciální predikáty s vyhrazeným označením T a F. T je predikát pravdivý ve všech bodech uvažovaného prostoru. Odpovídající množinou je universum. F je predikát nepravdivý ve všech bodech uvažovaného prostoru a odpovídá mu množina prázdná.

Předpokládejme výpočetní mechanismus (dále jen mechanismus) označený S a podmínku R, kterou musí splňovat stav mechanismu po skončení své aktivity. Podmínku R nazvěme "konečná podmínka" (angl. "postcondition"). Pak zápis wp(S,R) bude označovat nejslabší počáteční podmínku (angl. "weakest precondition"), která zaručuje, že mechanismus se dostane v konečné době do stavu splňujícího konečnou podmínku R. Není-li nejslabší počáteční podmínka splněna, nelze zaručit, že se mechanismus S dostane do stavu splňujícího R, i když to nesplnění podmínky nevylučuje. Při nesplnění wp(S,R) se může mechanismus dostat do stavu nesplňujícího R nebo do stavu nekonečné aktivity. Situace znázorňuje obr. 13.1.



Obr. 13.1. Znázornění výsledku aktivity S ve vztahu k nejslabší počáteční podmínce

Množina všech možných konečných podmínek pro daný mechanismus je tak rozsáhlá, že její znalost např. v tabelární formě, která by umožnila rychlé určení wp(S,R) je prakticky nezvládnutelná. Proto je definice sémantiky mechanismu daná ve formě pravidel, popisujících, jak odvodíme k dané konečné podmínce R odpovídající nejslabší počáteční podmínku wp(S,R). Pro daný mechanismus S a daný predikát R je pravidlo, jež dá za výsledek wp(S,R) označováno jako "transformace predikátů" a definuje se jí sémantika mechanismu S.

Nejčastěji nás však nezajímá úplná sémantika mechanismu. Mechanismu S používáme pouze pro zvláštní účel - pro ustavení pravdivosti určité konečné podmínky R, pro niž byl mechanismus navržen. Ani pro tuto určitou konečnou podmínku R nás nezajímá přesná a úplná forma wp(S,R), ale obvykle o něco silnější "postačující" podmínka P, pro niž platí

$$P \Rightarrow wp(S,R) \text{ pro všechny stavy.}$$

Pak P je postačující počáteční podmínka. V terminologii množin to znamená, že množina stavů označená P je podmnožinou množiny stavů označené wp(S,R).

Chápeme-li transformaci predikátu wp(S,R) jako funkci konečné podmínky R, pak má tato funkce několik základních vlastností:

1) Pro každý mechanismus S platí $wp(S,F)=F$ (2.1)
   Této vlastnosti se také říká "zákon vyloučeného zázraku".

2) Pro každý mechanismus S a konečné podmínky Q a R takové, že platí $Q \Rightarrow R$ pro všechny stavy,
   platí také $wp(S,Q) \Rightarrow wp(S,R)$ pro všechny stavy. (2.2)
   Této vlastnosti říkáme zákon monotónnosti.

3) Pro každý mechanismus S a konečné podmínky Q a R platí :
   $wp(S,Q) \wedge wp(S,R) = wp(S,Q \wedge R)$ pro všechny stavy (2.3)
   a také
   $wp(S,Q) \vee wp(S,R) = wp(S,Q \vee R)$ pro všechny stavy (2.4)

13.3. Definice základních mechanismů

Definujme pro tvorbu algoritmů tyto elementární mechanismy :

1) Prázdný příkaz "skip", jehož sémantika je dána transformací
   wp("skip",R)=R (3.1)

2) Příkaz zastavení v důsledku chybového stavu "abort", kde
   wp("abort",R)=F (3.2)

3) Přiřazovací příkaz

$\underline{wp("x:=E",R)=R^x_E}$ (3.3)

kde zápisem R^x_E se rozumí textová kopie R, v níž je každý výskyt proměnné x nahrazen výrazem E.

např, wp("x:=7", x=7) = (7=7) = T

nebo wp("x:=7", x=6) = (7=6) = F

nebo wp("x:=x-1", $x^2 \geq 1$) = $((x-1)^2 \geq 1) = (x \geq 2 \vee x \leq \emptyset)$ =

= $(x \neq 1)$ {pro celá čísla}

Pomocí BNF lze příkaz definovat takto :

⟨příkaz⟩ ::= "skip" | "abort" | ⟨přiřazovací příkaz⟩

⟨přiřazovací příkaz⟩ ::= ⟨proměnná⟩ := ⟨výraz⟩

Pro některé účely rozšíříme přiřazovací příkaz o možnost paralelního přiřazení takto :

⟨přiřazovací příkaz⟩ ::= ⟨proměnná⟩ := ⟨výraz⟩ | ⟨proměnná⟩, ⟨přiřazovací příkaz⟩ , ⟨výraz⟩

Tento příkaz umožní např. zápisem <u>X1,X2:=E1,E2</u> přiřadit dvěma proměnným současně hodnoty dvou výrazů nebo zápisem <u>X,Y:=Y,X</u> provést vzájemnou výměnu hodnot dvou proměnných.

13.4. Definice složených příkazů

Nejjednodušším způsobem, jak odvodit ze dvou daných funkcí jednu funkci novou je způsob, v němž hodnota první funkce slouží jako argument druhé. Již tradičně má notace takového složení tvar "S1;S2" a jeho sémantika je dána vztahem <u>wp("S1;S2",R)= wp(S1, wp(S2,R))</u> (4.1)

Tato definice se často interpretuje jako sémantická definice středníku. Jinými slovy říká: jestliže v posloupnosti "S1;S2" má mechanismus S2 dosáhnout konečného stavu splňujícího podmínku R, pak jeho nejslabší počáteční podmínku musí zaručit konečný stav mechanismu S1. Jeho nejslabší počáteční podmínka je tedy rovna nejslabší počáteční podmínce mechanismu "S1;S2" k dosažení stavu splňujícího R.

Příklad: Příkazy "x:=x+y;y:=x-y;x:=x-y" realizují vzájemnou výměnu hodnot proměnných x a y tedy "x,y:=y,x"

Důkaz: Dosaďme do vztahu (4.1) a s pomocí (3.3) dostaneme :

wp("x:=x+y;y:=x-y;x:=x-y", $x=X \wedge y=Y$) =

= wp("x:=x+y;y:=x-y",wp("x:=x-y", $x=X \wedge y=Y$)) =

= wp("x:=x+y;y:=x-y", $x-y=X \wedge y=Y$) =

= wp("x:=x+y",wp("y:=x-y", $x-y=X \wedge y=Y$)) =

= wp("x:=x+y", $x-(x-y)=X \wedge (x-y)=Y$) =

= wp("x:=x+y", $y=X \wedge (x-y)=Y$) =

= $y=X \wedge (x+y)-y=Y$ =

= $\underline{x=X \wedge x=Y}$ Q.E.D.

Složitější kompozicí jednoduchých příkazů jsou <u>řízené příkazy</u>. Umožňují tvorbu alternativních a repetičních řídicích struktur. Definici příkazu pak můžeme pomocí BNF rozšířit o alternativní příkaz "IF" a repetiční příkaz "DO" takto :

⟨příkaz⟩ ::= ... | <u>if</u> ⟨soubor řízených příkazů⟩ <u>fi</u> |
<u>do</u> ⟨soubor řízených příkazů⟩ <u>od</u>

⟨soubor řízených příkazů⟩ ::= ⟨řízený příkaz⟩ {§ ⟨řízený příkaz⟩}

⟨řízený příkaz⟩ ::= ⟨řídicí hlavička příkazu⟩ { ; ⟨příkaz⟩}

⟨řídicí hlavička příkazu⟩ ::= ⟨Booleovský výraz⟩→⟨příkaz⟩

kde symbol § (v orig. byl použit zvláštní znak ve tvaru "stojatého obdélníčku" ▯) má funkci oddělovače jednotlivých alternativ, jejichž pořadí v souboru nemá žádný význam.

13.4.1. Popis příkazu "IF"

Alternativní příkaz "IF" má několik důležitých vlastností:

a) Předpokládá se, že všechny řídicí Booleovské výrazy jsou definované. V jiném případě může vyhodnocení nedefinovaného výrazu vést k nesprávně provedené aktivitě a tedy i celý příkaz "IF" nemusí pracovat správně.

b) Obecně vede řídicí struktura "IF" k nedeterminovanosti, protože pro každý počáteční stav, který způsobí, že více než jeden řídicí Booleovský výraz je pravdivý, může být pro určení aktivity vybrán kterýkoli z nich.

c) Jestliže je počáteční stav takový, že žádný z Booleovských řídicích výrazů není pravdivý, pak aktivace takového počátečního stavu povede k zastavení s chybou a v tom případě je řídicí struktura "IF" ekvivalentní příkazu "abort". K témuž vede i příkaz "IF" s prázdným souborem řízených příkazů, tedy konstrukce "<u>if</u> <u>fi</u>".

Nejslabší počáteční podmínka příkazu "IF" je stanovena takto : Nechť "IF" je označení příkazu, jehož tvar je

$$\underline{\text{if}}\ B_1 \rightarrow S_1\ \S\ B_2 \rightarrow S_2\ \S\ \ldots\ \S\ B_n \rightarrow S_n\ \underline{\text{fi}}$$

kde S_i je seznam příkazů řízených výrazem B_i, pak pro libovolnou konečnou podmínku R platí :

$$\underline{wp(\text{"IF"},R)=(\underline{E}\ j:1\le j\le n:B_j)\wedge(\underline{A}\ j:1\le j\le n:B_j \Rightarrow wp(S_j,F))} \qquad (4.1.1)$$

(Symbol <u>E</u> bude používán místo existenčního kvantifikátoru "$\exists$", a symbol <u>A</u> místo všeobecného kvantifikátoru "$\forall$").

13.4.2. Popis příkazu "DO"

Formální definice nejslabší počáteční podmínky pro repetiční příkaz "DO" je poněkud složitější než pro příkaz "IF".
Nechť "DO" je označení příkazu, jehož tvar je

$$\underline{\text{do}}\ B_1 \rightarrow S_1\ \S\ B_2 \rightarrow S_2\ \S\ \ldots\ \S\ B_n \rightarrow S_n\ \underline{\text{od}}$$

a nechť "IF" je označení alternativního příkazu se stejným souborem řízených příkazů. Nechť podmínky $H_k(R)$ jsou definovány takto :

$$H_\emptyset(R)=R\wedge\neg(\underline{E}\ j:1\le j\le n:\ B_j) \qquad (4.2.1)$$

a pro $k>\emptyset$: $$H_k(R)=wp(\text{"IF"},H_{k-1}(R))\vee H_\emptyset(R) \qquad (4.2.2)$$

pak $$wp(\text{"DO"},R)=(\underline{E}\ k:k\ge\emptyset:\ H_k(R)) \qquad (4.2.3)$$

Intuitivně je $H_k(R)$ nejslabší počáteční podmínka zabezpečující ukončení aktivity příkazu "DO" po maximálně k "průchodech". Každý průchod je určen výběrem některého z řídicích výrazů a aktivuje odpovídající řízené příkazy. $H_k(R)$ současně zabezpečuje, že po ukončení příkazu "DO" bude systém ve stavu splňujícím konečnou podmínku R.

Pro $k=\emptyset$ ukončí příkaz "DO" svou aktivitu, aniž provede výběr některého řídicího výrazu, protože žádný z nich, jak plyne z (4.2.1) není pravdivý. Pak počáteční pravdivost podmínky R je nutnou a postačující podmínkou pro splnění konečné podmínky R.

Pro $k > \emptyset$ rozlišíme dva případy :

a) Žádný z řídicích výrazů není pravdivý a v tom případě z (4.2.1) a (4.2.2) plyne pravdivost R

b) Alespoň jeden řídicí příkaz je pravdivý. Pak se provede jeden průchod, který je ekvivalentní příkazu "IF" se stejnou strukturou řízených příkazů. (Protože druhý člen pravé strany vztahu (4.2.1) je nepravdivý, nemůže dojít k "abortu"). Po skončení aktivity příkazů tohoto průchodu musí přejít systém do stavu, který zabezpečuje, že po maximálně k-1 dalších průchodech bude ustaven stav splňující podmínku $H_{\emptyset}(R)$. Je to zabezpečeno tím, že konečným stavem průchodu (resp. ekvivalentního příkazu "IF") je podle (4.2.2) podmínka $H_{k-1}(R)$.

Repetiční příkaz "DO", jehož počáteční stav splňuje podmínku (4.2.1) je ekvivalentní prázdnému příkazu "skip". K témuž vede i prázdný soubor řízených příkazů, tedy konstrukce do od.

13.5. Teorém alternativního příkazu "IF"

Nechť je dán příkaz "IF" a predikát BB pro nějž platí :

$$BB = (\underline{E}\ j: 1 \le j \le n:\ B_j)$$

S použitím uvedených konvencí má teorém alternativního příkazu tento tvar :
Nechť P a Q jsou predikáty pro něž platí :

$$\underline{(\underline{A}\ j: 1 \le j \le n:\ (P \wedge B_j) \Rightarrow wp(S_j, Q))} \qquad (5.1)$$

a také $\qquad \underline{P \Rightarrow BB} \qquad (5.2)$

pak $\qquad \underline{P \Rightarrow wp(\text{"IF"}, Q)} \qquad (5.3)$

Důkaz : Podle definice příkazu "IF" (4.1.1) platí

$$wp(\text{"IF"}, Q) = (\underline{E}\ j: 1 \le j \le n:\ B_j) \wedge (\underline{A}\ j: 1 \le j \le n:\ B_j \Rightarrow wp(S_j, Q))$$

Musíme tedy dokázat

$$P \Rightarrow (\underline{E}\ j: 1 \le j \le n:\ B_j) \wedge (\underline{A}\ j: 1 \le j \le n:\ B_j \Rightarrow wp(S_j, Q))$$

Z (5.2) vyplývá implikace prvního členu pravé strany a stačí tedy dokázat, že pro všechny stavy platí :

$$\underline{P \Rightarrow \underline{A}\ j: 1 \le j \le n:\ B_j \Rightarrow wp(S_j, Q)} \qquad (5.4)$$

Pro všechny stavy, pro něž je P nepravdivé, je vztah (5.4) pravdivý z definice implikace. Všechny stavy, pro něž je P pravdivé a pro všechna j rozlišíme dva případy :

a) Buď je B_j nepravdivé, pak je $B_j \Rightarrow wp(S_j, Q)$ pravdivé z definice implikace a pak je pravdivý i vztah (5.4)

b) nebo je B_j pravdivé a pak je na základě (5.1) pravdivé i $wp(S_j, Q)$. Pravdivé je tedy i $B_j \Rightarrow wp(S_j, Q)$ a tím i (5.4).

Tím je dokázána platnost vztahu (5.4) a tudíž i (5.3) Q.E.D.

Závěr teorému říká, že P, které splňuje podmínky (5.1) a (5.2), implikuje nejslabší počáteční podmínku zabezpečující počáteční stav, který se mechanismem "IF" změní do konečného stavu, splňujícího podmínku Q. Odvození vyhovujícího P je tedy důkazem správnosti alternativní konstrukce "IF". Skutečnost, že premisy mají stejný tvar jako závěr je zárukou, že důkaz konstrukce bude mít, vzhledem ke konstrukci samotné, lineární charakter co do své délky.

13.6. Teorém invariance pro repetiční příkaz "DO"

Tento teorém, který poprvé odvodil C.A.R.Hoare, se považuje za jeden z nejzávažnějších teorémů programování.

Zaveďme pomocné formální prostředky :
Zápis wdec(S,t) nechť se interpretuje jako nejslabší počáteční podmínka pro takový počáteční stav, který zaručuje, že mechanismus S v konečné době sníží hodnotu t, kde t je celočíselná funkce proměnných programu. Pak lze psát :

$$wdec(S,t)=wp("\tau:=t;S",\ t<\tau) \qquad (6.1)$$

Tento vztah lze podle (4.1) rozvést na:
$wdec(S,t)=wp("\tau:=t",\ wp(S,t<\tau))$ a konečně na

$$wdec(S,t)=[wp(S,t<\tau)]^{\tau}_{t} \qquad (6.2)$$

kde pravá strana (6.2) se interpretuje tak, že ve výrazu konečné podmínky bude každé τ nahrazeno t. Použití wdec(S,t) ilustruje následující příklad.
Příklad : wdec("x:=x-y,x+y) je nejslabší podmínka, za níž příkaz "x:=x-y" sníží hodnotu funkce "x+y".
Podle (5.6.2)

$$wdec("x:=x-y",x+y)=[wp("x:=x-y",x+y<\tau)]^{\tau}_{x+y}=[x-y+y<\tau]^{\tau}_{x+y}=$$
$$=x<x+y=y>\emptyset$$

Je-li $y>\emptyset$, pak příkaz "x:=x-y sníží hodnotu funkce "x+y".
Nechť je repetiční příkaz "DO" definován zápisem
$\underline{do}\ B_1\rightarrow S_1\ \S\ B_2\rightarrow S_2\ \S\ \dots\ \S\ B_n\rightarrow S_n\ \underline{od}$ a predikát BB je definován
$BB=(\underline{E}\ j:1\leq j\leq n:\ B_j)$
Pak lze teorém invariance repetičního příkazu formulovat takto :
Nechť P je predikát takový, že platí

$$(\underline{A}\ j:1\leq j\leq n:\ (P\wedge B_j)\Rightarrow (wp(S_j,P)\wedge wdec(S_j,t))) \qquad (6.3)$$

a současně $P\Rightarrow t>\emptyset$ (6.4)
pak $P\Rightarrow wp("DO",P\wedge\neg BB)$ (6.5)

Tvrzení (6.5) říká, že jestliže počáteční stav zaručuje pravdivost predikátu P a jestliže je kterýkoliv z řídicích výrazů vybrán a jeho řízené příkazy provedeny, pak po skončení jejich aktivity zůstává predikát P pravdivý. Predikát P tedy zůstává pravdivý (invariantní) bez ohledu na počet, kolikrát budou ten či onen řídicí výraz a jeho řízené příkazy vybrány. Po skončení aktivity celé repetiční konstrukce "DO", kdy už žádný z řídicích výrazů není pravdivý, zaručuje konečný stav pravdivost tvrzení

$$P\wedge\neg BB$$

Vztah (6.3) zaručuje neustálé snižování funkce t a (6.4) zaručuje, že jeho hodnota je kladná. Funkce t je celočíselná funkce a je tedy spolehlivým prostředkem zakončení aktivity repetiční konstrukce.

Závažnost teorému invariance pro cyklus spočívá v tom, že pravdivost jeho výroků nezávisí na počtu průchodů. Z toho vyplývá, že lze postavit o cyklu tvrzení i v tom případě, kdy počáteční stav neurčuje tento počet průchodů. Umožňuje to provést důkaz správnosti repetiční konstrukce, jejíž délka není úměrná počtu průchodů cyklu.

13.7. Příklady

V této části kapitoly bude uvedeno několik příkladů známých problémů řešených za použití zavedených jazykových prostředků a metodiky důkazu vytvářeného programu. Příklady jsou zaměřeny především na práci s cyklem.

13.7.1. Největší společný dělitel dvou celých čísel

Nechť X,Y jsou celá čísla větší než nula. Hledáme největší společný dělitel (dále jen NSD) čísel X,Y. Řešení má tedy formálně tvar :

$$\underline{R : Z=NSD(X,Y) \wedge X>\emptyset \wedge Y>\emptyset} \qquad (7.1.1)$$

Z definice NSD dvou celých čísel vyplývá

$$\underline{NSD(X,Y)=NSD(Y,X)} \qquad (7.1.2)$$

a $\underline{NSD(X,X)=X}$ (7.1.3)

Lze dokázat, že platí také

$$\underline{NSD(X,Y)=NSD(X-Y,Y) \wedge X>Y} \qquad (7.1.4)$$

a z toho lze odvodit i

$$\underline{NSD(X,Y)=NSD(X,Y-X) \wedge Y<X} \qquad (7.1.5)$$

$$\underline{NSD(X,Y)=NSD(X+Y,Y)=NSD(X,Y+X)} \qquad (7.1.6)$$

Pro získání řešení je důležitý vztah (7.1.3) a jestliže konečný stav mechanismu zajistí relaci x=y, pak tento stav zajišťuje také řešení $\underline{NSD(x,y)=x}$. Mechanismus však musí také zajistit neměnnost (invarianci) relace:

$$\underline{P : NSD(X,Y)=NSD(x,y) \wedge \emptyset<x\leq X \wedge \emptyset<y\leq Y} \qquad (7.1.7)$$

Z počátečního stavu x=X a y=Y bude mechanismus zpracovávat x a y tak, aby se zachovala invariance P s cílem dosáhnout relace x=y. Vztah (7.1.4) resp. (7.1.5) nabízí změnu x a y jejich rozdílem. Prozkoumejme podmínku, ze níž příkaz "x:=x-y' dosáhne žádoucí konečné podmínky P.

$$\underline{wp(\text{"x:=x-y"},P)=(NSD(x-y,y)=NSD(X,Y) \wedge \emptyset< x-y\leq X \wedge \emptyset<y\leq Y} \qquad (7.1.8)$$

Z teorému invariance pro cyklus vyplývá, že najdeme-li P a B_i takové, že platí $(P\wedge B_i) \Rightarrow wp(S_i,P) \wedge wdec(S_i,t)$ a $P\Rightarrow t>\emptyset$ pak $P\Rightarrow wp(\text{"DO"},P\wedge\neg BB)$.
Z (7.1.7) je vidět, že P implikuje všechny členy pravé strany vztahu (7.1.8) s výjimkou $(\emptyset< x-y)$. Z toho vyplývá, že :

$$\underline{(P\wedge x>y)\Rightarrow wp(\text{"x:=x-y"},P)} \qquad (7.1.9)$$

a v důsledku symetrie také :

$$\underline{(P\wedge y>x)\Rightarrow wp(\text{"y:=y-x"},P)} \qquad (7.1.10)$$

Zbývá najít funkci t, která vyhovuje podmínkám teorému invariance. Nechť t=x+y.
Z (7.1.7) platí $P\Rightarrow t>\emptyset$, a pak tedy

$$wdec(\text{"x:=x-y"},x+y)= \left[wp(\text{"x:=x-y"},\tau>x+y)\right]^{\tau}_{t} =$$
$$=\left[\tau>(x-y)+y\right]^{\tau}_{x+y} = (x+y>x) = \underline{y>\emptyset}$$

a pak tedy platí z (7.1.7), že $P\Rightarrow wdec(\text{"x:=x-y"},x+y)$.

Výsledný program má tuto strukturu :
"Ustav počáteční podmínku P";
<u>do</u> "Snižuj t za invariance P"
<u>od</u>; $\{P\wedge\neg B\Rightarrow R\}$

Jeho konečný tvar je :

x:=X; y:=Y; $\{P\}$
<u>do</u>
  x > y   x:=x-y $\{P\wedge B_1\}$
§ y > x   y:=y-x $\{P\wedge B_2\}$

od;
z:=x; $\{P \wedge \neg BB)\}$ $\{R\}$

13.7.2. Součin dvou celých kladných čísel

Nechť X,Y jsou celá kladná čísla $X>0 \wedge Y>0$.
Pak řešení součinu čísel X a Y má tvar :

$R : Z * Y$ (7.2.1)

Často se invariantní relace cyklu tvoří pomocí pomocných proměnných, které na počátku nabývají hodnot argumentů. Jejich změnou v průběhu cyklu při zachování invariance relace dosáhneme řešení. Pak vhodnou relací je :

$P : z+x*y=X*Y \wedge 0 \leq x \leq X \wedge 0 \leq y \leq Y$ (7.2.2)

Z (7.2.1) a (7.2.2) vyplývá, že $(P \wedge y=0) \Rightarrow R$ (7.2.3)

Program pak bude mít formální tvar

"Ustavení počátečního stavu splňujícího P";
do y≠0 → "snižování hodnoty y při zachování P"
od; $\{P \wedge y=0\}$ $\{R\}$

Nejjednodušším mechanismem snižujícím hodnoty y a současně zabezpečujícím konečnost cyklu je příkaz "y:=y-1".

Pak $wp(\text{"y:=y-1"},P)=(z+x*(y-1)=X*Y \wedge 0 \leq x \leq X \wedge 0 \leq y-1 \leq Y) =$
$=(z-x+y*y=X*Y \wedge 0 \leq x \leq X \wedge 0 \leq y-1 \leq Y)$ (7.2.4)

Aby se zachovala invariance P, je třeba současně s přiřazením y:=y-1 provést přiřazení z:=z+x, které kompenzuje v (7.2.4) úbytek vzniklý snížením y.

Program pak nabude tvaru

x,y,z:=X,Y,0; $\{P\}$
do
   y≠0 → z,y:=z+x,y-1 $\{P \wedge B_1\}$
od; $\{P \wedge \neg BB\}$ $\{R\}$

Snižování y lze provést rychleji, připustíme-li některé další operace. Nechť Booleovská operace "dělitelnost čísla a číslem b beze zbytku" má tvar "a./.b" a nechť operátorem pro celočíselné dělení je div. Pak za předpokladu, že y je sudé, tedy za předpokladu platnosti podmínky y./.2 (7.2.5)

lze snižování provést přiřazovacím příkazem "y:=y div 2" .

Aby se zachovala invariance P, je třeba současně kompenzovat tento úbytek y přírůstkem x pomocí příkazu "x:=x*2". Cyklus

do y./.2 → x,y:=x*2,y div 2 od

zachová invarianci P a současně končí neplatností (7.2.5), a to znamená, že y je nyní liché. Takový cyklus se s výhodou bude doplňovat s původním příkazem "y,z:=y-1,z+x", který při zachování P ustaví znova sudost proměnné y a tím podmínku pro jeho urychlené snižování. Program má pak tento konečný tvar :

x,y,z:=X,Y,0; $\{P\}$
do y≠0 → do y./.2 → x,y:=x*2,y div 2 od; $\{\neg y./.2\}$
   y,z:=y-1,z+x $\{y./.2\}$
od; $\{P \wedge \neg BB\}$ $\{R:z=X*Y\}$

13.7.3. Binární vyhledávání

Nechť je dáno pole celých čísel, pro která platí

$A[0] \leq A[1] \leq \dots \leq A[N-1] < A[N]$

a nechť je dáno $\underline{A[\emptyset] \leq x < A[N]}$

Nalezněme algoritmus, který ustaví pravdivost Booleovské proměnné EX v případě, že x se rovná hodnotě některého prvku zadaného pole, tedy

$$R\ :\ EX = (\underline{E}\ i : \emptyset \leq i < N : x = A[i]) \qquad (7.3.1)$$

Vzhledem k setříděnosti pole skončí repetiční proces dosažením podmínky

$$R'\ :\ A[i] \leq x < A[i+1] \qquad (7.3.2)$$

Pak platí

$$(R' \wedge EX = (x = A[i])) \Rightarrow R \qquad (7.3.3)$$

Invariantní relaci zavedeme pomocí proměnné j a vztahu (7.3.2) s cílem, aby $P \wedge (j = i+1) \Rightarrow R'$. Pak tedy

$$\underline{P\ :\ A[i] \leq x < A[j] \wedge \emptyset \leq i < j \leq N} \qquad (7.3.4)$$

Cílem cyklu je zpracovat hodnoty i a j při invarianci P tak, aby se dosáhlo platnosti relace j=i+1. Pak tedy program bude mít strukturu :

```
i,j:=Ø,N;   {P}
do j≠i+1    "zpracování i a j při zachování P"
od;  {P ∧ j=i+1}   {R'}
EX:=(x=A [i]);     {R}
```

Nechť v průběhu zpracování nabude proměnná i resp. j nové hodnoty $\underline{m}$. Nalezněme nejslabší počáteční podmínky pro mechanismus "i:=m" resp. "j:=m" :

$$\underline{wp(\text{"i:=m"}, P) = A[m] \leq x < A[j] \wedge \emptyset \leq m \leq j \leq N \;=\; P \wedge A[m] \leq x \wedge m < j} \qquad (7.3.5)$$

$$\underline{wp(\text{"j:=m"}, P) = A[i] \leq x < A[m] \wedge \emptyset \leq i < m \leq N \;=\; P \wedge x < A[m] \wedge i < m} \qquad (7.3.6)$$

Nechť t=j-i. Nalezněme podmínky, za nichž zvolené mechanismy zaručí konečnost aktivity cyklu.

$$wdec(\text{"i:=m"}, \tau > j-i) = [\tau > j-m]^{\tau}_{j-i} \;=\; j-i > j-m \;=\; m > i \qquad (7.3.7)$$

$$wdec(\text{"j:=m"}, \tau > j-i) = [\tau > m-i]^{\tau}_{j-i} \;=\; j-i > m-i \;=\; j > m \qquad (7.3.8)$$

Jak vyplývá ze (7.3. - 5,6,7 a 8), musí nová hodnota m splňovat podmínku

$$\underline{i < m < j} \qquad (7.3.9)$$

Z hlediska symetrie je pro m vhodnou hodnota půlící interval (i,j)

m:=(i+j) <u>div</u> 2.

Vztah $P \wedge j \neq i+1$, který platí po celou dobu cyklu, zajišťuje pro toto m platnost (7.3.9). Protože $i_{max} = j-2$ pak

(i+j) <u>div</u> 2 = (2*j-2) <u>div</u> 2 = j-1

a také $j_{min} = i+2$, a pak

(i+j) <u>div</u> 2 = (2*i+2) <u>div</u> 2 = i+1

Program pak bude mít tento konečný tvar :

```
i,j:=Ø,N;   {P}
do j≠i+1 → m:=(i+J) div 2;     {i < m < j}
          if A[m] ≤ x → i:=m      {(P ∧ A[m] ≤ x ∧ m < j) viz. (7.3.5)}
           § x < A [m] → j:=m     {(P ∧ x < A[m] ∧ i < m) viz. (7.3.6)}
          fi
od;  {P ∧ j=i+1}  {R'}
EX:=(x=A[i])      {R}
```

13.7.4. Teorém pro lineární vyhledávání

Nechť B je Booleovská funkce celočíselného algoritmu. Mějme program tvaru :

```
i:=Ø;
do B(i) → i:=i+1 od
```

Pro tento program, jehož základem je cyklus, lze napsat invariantní relaci P ve tvaru $P(i)=(\underline{A}\ j:\emptyset \leq j < i: \neg B(j))$ (7.4.1)

Důkaz: $wp(i:=i+1", P(i)) = P(i+1) =$

$=(\underline{A}\ j:\emptyset \leq j < i+1: \neg B(j)) = (\underline{A}\ j:\emptyset \leq j < i: \neg B(j)) \wedge \neg B(i) =$

$= \underline{P(i) \wedge \neg B(i)}$ Q.E.D

Uvedený cyklus se ukončí tehdy a jen tehdy, platí-li

$\underline{E}\ j:\emptyset \leq i: B(i)$ (7.4.2)

Pak bude platit

$P(i) \wedge B(i) = (\underline{A}\ j:\emptyset \leq j < i: \neg B(j)) \wedge B(i)$ (7.4.3)

což jinými slovy znamená, že $\underline{i}$ je nejmenší hodnota, pro niž platí podmínka B. Mechanismu, který vyhledává v zadané posloupnosti prvek s nejnižším pořadím, pro který platí zadaná podmínka, se říká <u>lineární vyhledávání</u>. Teorém pro lineární vyhledávání se uplatní v řadě problémů, jejichž součástí je právě lineární vyhledávání. Ilustrujme tento teorém na jednoduchém příkladě vyhledávání dané hodnoty v poli celých čísel.

Nechť je dáno pole celých čísel $A[\emptyset]$, $A[1]$, ..., $A[N-1]$ a nechť je dáno celé číslo x. Chceme stanovit řešení :

$R : EX=(\underline{E}\ i:\emptyset \leq i < N: x=A[i])$ (7.4.4)

Podle teorému o lineárním vyhledávání bude aktivita cyklu konečná pouze při pravdivosti $(\underline{E}\ i:\emptyset \leq i < N: x=A[i])$. Tuto pravdivost můžeme zaručit rozšířením pole o prvek $A[N]=x$ a zavedením R' : $EX=(\underline{E}\ i:\emptyset \leq i \leq N: x=A[i]$ (7.4.5)

pak $(R'\ i \neq N) \Rightarrow R$ (7.4.6)

Výsledný program má tvar

```
i,A[N]:=Ø,x;
do  A[i]≠x → i:=i+1 od
EX:= i≠N
```

Tento algoritmus je známý pod názvem "rychlé lineární vyhledávání.

13.7.5. <u>Ekvivalence dvou kruhových seznamů v polích</u>

Zjištění ekvivalence dvou kruhových seznamů je poměrně známý úkol z oblasti problémů označované jako "pattern recognition".

Nechť A a B jsou dva kruhové seznamy, oba o N prvcích. Úkolem je prověřit, zda jsou oba seznamy shodné, bez ohledu na případnou rotaci jednoho seznamu, nutnou k dosažení shody.

Nechť A_i je posloupnost $A_i=(A[i], A[i+1], ..., A[i+N-1])$, kde o každém indexu předpokládáme redukci pomocí operace modulo N. Pak řešení R má tvar

$R : b=(\underline{E}\ k,m:\emptyset \leq k \leq N, \emptyset \leq m \leq N: A_k=B_m)$ (7.5.1)

<u>Poznámka</u>: Vytvoříme-li množinu všech posloupností A_i a množinu všech posloupností B_i pro $i : \emptyset \leq i < N$, pak tyto dvě množiny jsou ekvivalentní při pravdivém výsledku úlohy.

<u>Úvaha</u> : Existuje nějaká reprezentativní posloupnost AA z množiny posloupností A_i a reprezentativní posloupnost BB z množiny posloupností B_i pro $i : \emptyset \leq i < N$, pro něž by platil vztah (7.5.2) ?

$R : b = (AA=BB)$ (7.5.2)

Vztah (7.5.2) platí např. pro AA resp BB, jež jsou lexikografickým minimem nebo lexikografickým maximem všech posloupností $A_\emptyset$ až A_{N-1} resp. $B_\emptyset$ až B_{N-1}. Nalezení lexikografických minim či maxim vede k řešení.

Yossi Shiolah [2] však nalezl výrazně efektivnější řešení. Jeho úvaha vychází z předpokladu, že nelze ustavit R bez jakéhokoli pokusu o porovnání posloupností A_i a B_j. Pak lze stanovit invariantní relaci pro cyklický charakter algoritmu :

$$P(i,j,k) : \emptyset \leq k \leq N \wedge (\underline{A}\ h:\emptyset \leq h < k: A[i+h]=B[j+h]) \qquad (7.5.3)$$

zřejmě platí:

$$_{\vee}(P(i,j,k) \wedge k=N \wedge b) \Rightarrow R \qquad (7.5.4)$$

Vztah

$$P(i,j,\emptyset)=T \qquad (7.5.5)$$

vyplývá z definice všeobecného kvantifikátoru pro všechna i a j. Jestliže pro jistou dvojici (i,j) je dosažení stavu k=N nemožné, pak zřejmě proto, že pro jisté k platí

$$\underline{E}\ k:\emptyset \leq k < N: A[i+k] \neq B[j+k]$$

Pak tedy platí tento vztah o lexikografické relaci:

$$(P(i,j,k) \wedge A[i+k]>B[j+k]) \Rightarrow (\underline{A}\ h:\emptyset \leq h \leq k: A_{i+h} \succ B_{j+h} \succeq BB) \qquad (7.5.6)$$

kde BB resp. AA je lexikografické minimum.

Vztah lze ilustrovat na malém příkladu :

		i	i+1				i+k	
A :	...	7	5	1	2	2	3	...
B :	...	7	5	1	2	2	1	...
		j	j+1				j+k	

Protože A[i+k] B[j+k], pak zřejmě platí i

$$A_i \succ B_j \wedge A_{i+1} \succ B_{j+1} \wedge \dots \wedge A_{i+k} \succ B_{j+k}$$

Z (7.5.6) tedy plyne :

$$(P(i,j,k) \wedge A[i+k]>B[j+k]) \Rightarrow (\underline{A}\ h:\emptyset \leq h \leq i+k+1: A_h \succ BB) \qquad (7.5.7a)$$

a symetricky také

$$(P(i,j,k) \wedge B[j+k] > A[j+k]) \Rightarrow (\underline{A}\ h:\emptyset \leq h \leq j+k+1: B_h \succ AA) \qquad (7.5.7b)$$

Zvolme pro pravý operand vztahu (7.5.7) označení

$$QA(i) : (\underline{A}\ h:\emptyset \leq h < i: A_h \succ BB) \qquad (7.5.8a)$$

a

$$QB(j) : (\underline{A}\ h:\emptyset \leq h < j: B_h \succ AA) \qquad (7.5.8b)$$

Pak z (7.5.7) a (7.5.8) vyplývá :

$$(P(i,j,k) \wedge QA(i) \wedge A[i+k] > B[j+k]) \Rightarrow QA(i+k+1) \qquad (7.5.9a)$$

a

$$(P(i,j,k) \wedge QB(j) \wedge B[j+k] > A[i+k]) \Rightarrow QB(j+k+1) \qquad (7.5.9b)$$

Vztahy (7.5.9) jsou podstatou vysoké efektivnosti výsledného algoritmu. K řešení R lze zřejmě ze vztahů (7.5.8) dospět splněním vztahů :

$$(QA(i) \wedge i=N \wedge \neg b) \Rightarrow R \qquad (7.5.10a)$$

a

$$(QB(j) \wedge j=N \wedge \neg b) \Rightarrow R \qquad (7.5.10b)$$

Invariantní relaci ze vztahu (7.5.3) nutno tedy rozšířit :

$$Z : P(i,j,k) \wedge QA(i) \wedge QB(j) \qquad (7.5.11)$$

Konečný tvar programu podle algoritmu YOSSI SHIOLAHA pak je :

```
i,j,k:=∅,∅,∅;
do
   k≠N ∧ i< N ∧ j< N        {jak plyne ze (7.5.4) a (7.5.10a,b)}
```

```
                    if A[i+k] =B [j+k] --> k:=k+1
                     § A[i+k] >B [j+k] --> i:=i+k+1; k:=Ø
                     § B [j+k] >A [i+k] --> j:=j+k+1; k:=Ø
                    fi
od;
b:=k=N
```

Celková efektivnost algoritmu, která při tradičním pojetí dvou cyklů je úměrná kvadrátu počtu prvků seznamu, je v tomto případě <u>v nejhorším úměrná vztahu</u> $\underline{k+i+j \leq 3N-1}$!

Dále je uvedena verze tohoto programu zapsaná v Pascalu ADT :

```
PAGE     1 * FEL-PASCAL * ADT-RTF-82/11 *  5:05 PM  TUE., 20  DEC., 1983@@

00000  PROGRAM SEZNAMY(INPUT,OUTPUT);
00001  (*
00001     PROGRAMOVAL: K. SOLNICKY, III-EP-4
00001     KDE:         VUT BRNO
00001     KDY:         24. 6. 1983
00001
00001     PROGRAM NACTE DVA KRUHOVE SEZNAMY
00001     A SROVNAVA JE PODLE ALGORITMU
00001     YOSSI SHILOAM
00001  *)
00001
00001  CONST N=10;
00001
00001  TYPE
00001     TYPINDEXU=0..9;
00001     TYPPOLE=ARRAY[TYPINDEXU] OF INTEGER;
00001
00001  VAR
00001     I,J,L: INTEGER;
00001     A,B: TYPPOLE;
00001
00001  FUNCTION INDEX(IND: INTEGER): TYPINDEXU;
00036  (*
00036     TATO FUNKCE ZAJISTUJE ZMENU INDEXU TAK, ABY SE
00036     PRISLUSNYM POLEM PROCHAZELO JAKO KRUHOVYM SEZNAMEM
00036  *)
00036     BEGIN
00046        INDEX:=IND MOD N;
00053     END;  (* OF INDEX *)
00055
00055  BEGIN
00075
00075     (* NACTEME SEZNAMY *)
00075     WRITE(' PRVNI SEZNAM  ');
00103     FOR I:=0 TO N-1 DO BEGIN READ(A[I]); WRITE(A[I]:3); END;
00132     READLN; WRITELN;
00140     WRITE(' DRUHY SEZNAM  ');
00146     FOR I:=0 TO N-1 DO BEGIN READ(B[I]); WRITE(B[I]:3); END;
00175     WRITELN;
00200     I:=0;
00202     J:=0;
00204     L:=0;
00206
00206     (* ZACINAME SROVNAVAT,
00206        V L JE DELKA VE KTERE SE SEZNAMY SROVNALY *)
00206
00206     WHILE (L<>N)    (* SEZNAMY SE NESROVNALY *)
00211          AND (I<N) AND (J<N)    (* JESTE JE CO SROVNAVAT *)
00221     DO BEGIN
00221        IF A[INDEX(I+L)] = B[INDEX(J+L)] THEN L:=L+1;
00244        IF A[INDEX(I+L)] > B[INDEX(J+L)] THEN BEGIN
00266           I:=I+L+1;
00273           L:=0;
00275        END;
00275        IF A[INDEX(I+L)] < B[INDEX(J+L)] THEN BEGIN
00317           J:=J+L+1;
```

```
PAGE    2 * FFL-PASCAL * ADT-RTE-82/11 *  5:05 PM  TUE., 20  DEC., 1983@@

00324              L:=0;
00326           END;
00326        END;
00327        IF L=N THEN WRITELN(' SEZNAMY SE ROVNAJI')
00341               ELSE WRITELN(' SEZNAMY SE NEROVNAJI');
00352     END.

***       0 ERRORS
***    10.5 SECONDS
***     434 WORDS OF OBJECT CODE
```

Ukázka výpočtu pro dva jednoduché seznamy :

```
PRVNI SEZNAM    0  1  2  3  4  5  6  7  8  9
DRUHY SEZNAM    4  5  6  7  8  9  0  1  2  3
SEZNAMY SE ROVNAJI

PRVNI SEZNAM    1  2  3  4  5  6  7  8  9  0
DRUHY SEZNAM    1  1  2  3  4  5  6  7  8  9
SEZNAMY SE NEROVNAJI.
```

13.8. Z á v ě r

Uvedená metodika tvorby dokázaných programů, je prvním rozsáhlejším a systematickým pokusem ve velké a otevřené oblasti matematického důkazu programů. Zdaleka nevyčerpává všechny problémy, jež s dokazováním programů souvisí; nejsou např. vyřešeny metody pro formální práci se složitějšími datovými strukturami a s rekurzí. Pomocí této metodiky pracuje podle [2] několik desítek špičkových programátorů ve světě. Cílem jejich práce je, kromě výzkumu v této oblasti, také přetvoření známých základních algoritmů a vytvoření jakéhosi "katalogu" dokázaných a efektivních programů. Jak vyplývá z řady příkladů uvedených v tomto příspěvku, vede tvorba uvedené metodiky k přehledným, efektivním a z programátorského hlediska "pěkným programům. Potvrzuje to jedno z pravidel strukturovaného programování, jež tvrdí, že cesta ke zrychlení složitějšího programu nevede přes úspory času získané šamanskou optimalizací kódu daného algoritmu, ale spíše cestou hledání nového, efektivnějšího algoritmu.

Uvedený způsob tvorby programů není zřejmě nutný pro tvorbu programů na nízké a rutinní úrovni, ale zdá se být nepochybným, že vzdělaný programátor by měl být přinejmenším seznámen s metodikou tvorby dokázaných programů. Aplikace této metodiky může mít v budoucnu ve svých důsledcích i významný ekonomický přínos. Vede k tvorbě programů s vysokou spolehlivostí. Tvorba sama není náročná na technické prostředky a závěr práce - realizace programů je charakteristický minimalizací nároků na strojový čas počítače, protože tento čas by se měl redukovat o čas potřebný na odladění logických chyb vzniklých při tradičním návrhu a tvorbě algoritmů.

13.9. L i t e r a t u r a

[1] DIJKSTRA,E.W. : A Discipline of Programming Prentice Hall, 1976

[2] FEIJEN,W.H.J.: Seminář "A Discipline of Programming" konaný na PF UJEP v Brně ve dnech 10. - 14.12.1979