Pavol Jozef Šafárik University in Košice
Faculty of Science

Testovanie a verifikácia programov
Úvod

Gabriela Andrejková

Úvod

Čím zložitejšie a dôležitejšie úlohy zverujeme počítačom, tým pálčivejšou sa stáva otázka, či programy, podľa ktorých sú tieto úlohy riešené, sú správne.

S dôležitosťou programu spravidla rastie doba a náročnosť testovania programu, ktoré program musí podstúpiť pred odovzdaním na bežné používanie.

Samotné testovanie, alebo "ladenie"programu na overenie jeho správnosti nestačí.

Úvod

Z jeho samotnej podstaty vyplýva, že len jedinú odpoveď môže dať s plnou istotou: *program je nesprávny*. Ak totiž pre n rôznych vstupných dát vydal testovaný program správnu odpoveď, stále zostáva otvorená možnosť, že pri $(n+1)$-vých vstupných dátach bude v programe odhalená chyba.

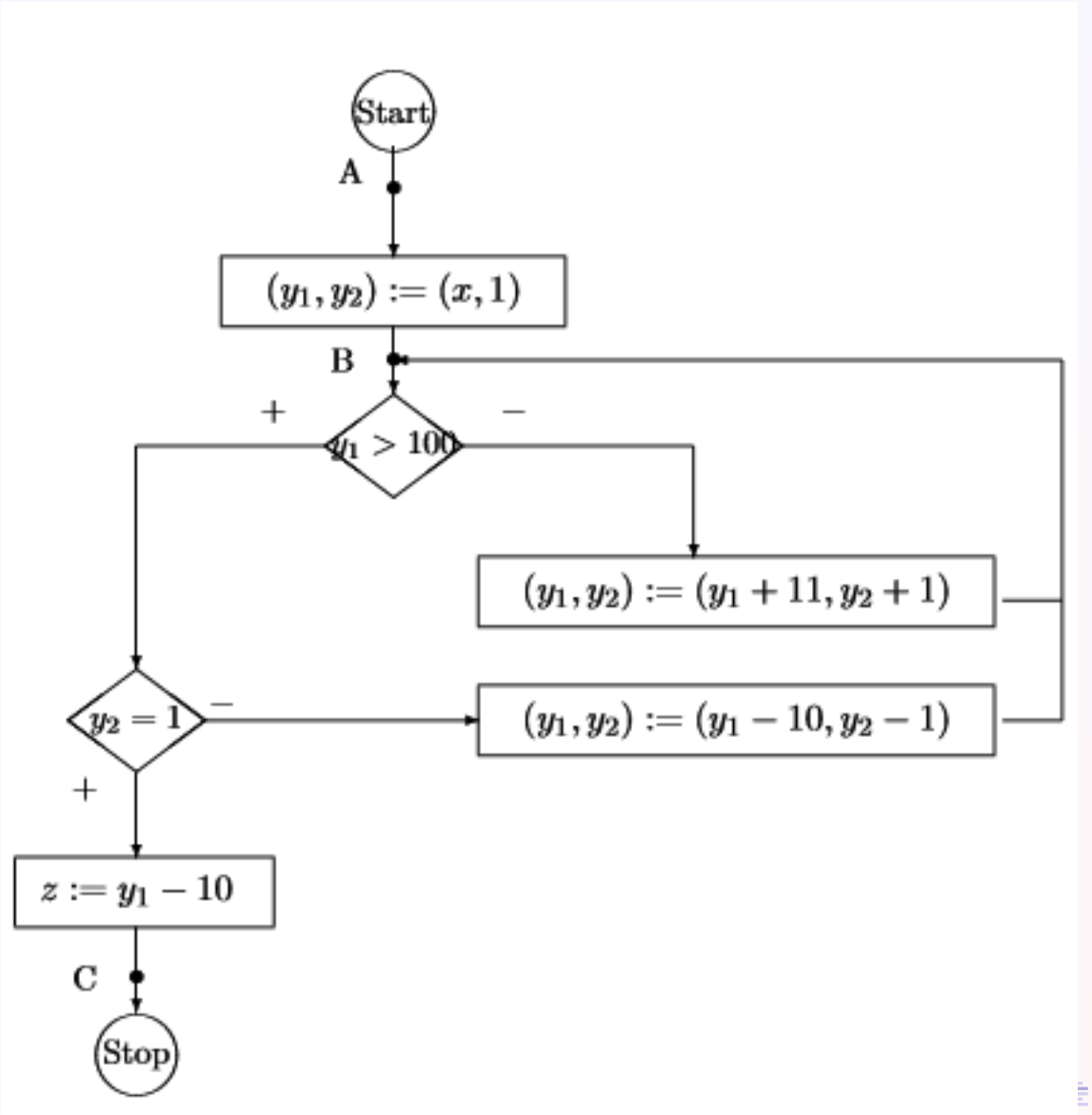
Skutočnú istotu o korektnosti programu môže poskytnúť len dôkaz jeho správnosti.

Úvod

Taký dôkaz potrebujeme, ak sa chceme presvedčiť, že program, ktorý sme vytvorili, je správny.

Potrebujeme ho ale aj vtedy, keď máme od iného programátora prevziať skutočne kľúčový úsek programového systému.

Ak sa chcete hneď teraz presvedčiť, že sa vyplatí niečo o dokazovaní správnosti programov vedieť, pozrite sa na nasledujúci program:


Start
A
$(y_1, y_2) := (x, 1)$
B
+
$y_1 > 100$
−
$(y_1, y_2) := (y_1 + 11, y_2 + 1)$
$y_2 = 1$
−
$(y_1, y_2) := (y_1 - 10, y_2 - 1)$
+
$z := y_1 - 10$
C
Stop

Jedna sa o vývojový diagram algoritmu pre výpočet tzv. **McCarthyho funkcie 91**.

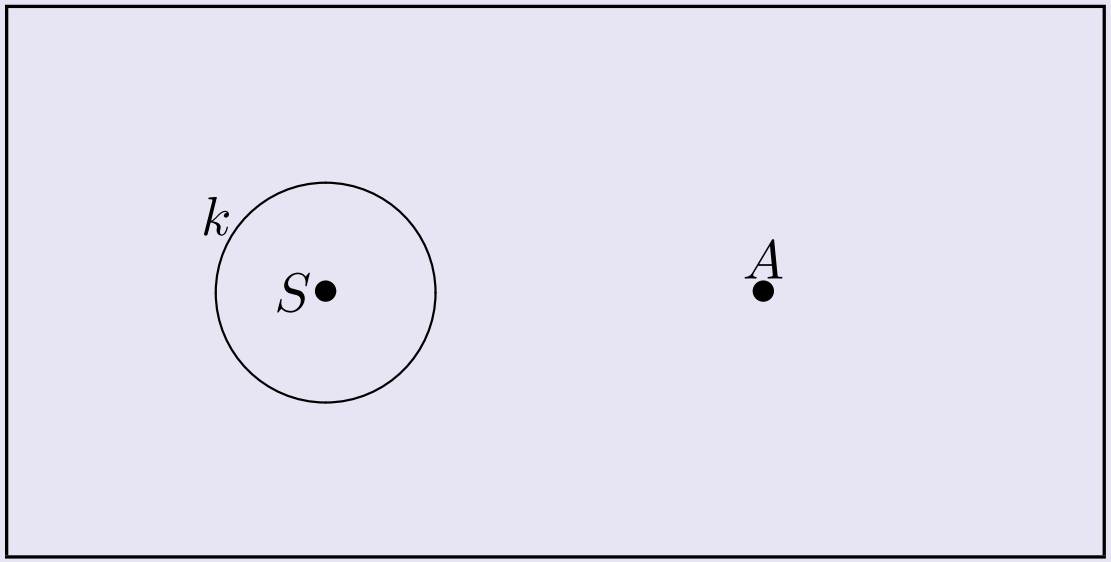
Ak pre vstupnú hodnotu $x, x \in N$, platí, $x > 100$, tak algoritmus vydáva na výstupe hodnotu $x - 10$, ak platí $0 \leq x \leq 100$, vydáva hodnotu 91.

Vyhovuje program uvedenému popisu pre každé x?

Podarilo sa vám nadobudnúť úplnú istotu?
Ako dlho vám to trvalo?
Ako dlho by vám trvalo vysvetliť to niektorému z kolegov?

Príklad:

Uveďte algoritmus, ktorým je možné pomocou kružidla a pravítka zostrojiť dotyčnicu ku kružnici prechádzajúcu daným bodom, obrázok 2.



Obr.: Zadanie k príkladu 1

Zostrojíme kružnicu kk s priemerom $|AS|$. Body B, C, v ktorých sa kružnice k a kk pretnú sú body dotyku hľadaných dotyčníc. Hľadaná dotyčnica je teda priamka AB (resp. AC).

Dôkaz správnosti tohto algoritmu poznáte zo školy: ak je B hľadaný bod dotyku, je trojuholník ASB pravouhlý, s pravým uhlom pri vrchole B. Vrcholy všetkých pravouhlých trojuholníkov ležia na Thalesovej kružnici nad úsečkou AS, atď.

Niekoľko rysov dokazovania správnosti algoritmov

Ponúka sa toto pozorovanie: akonáhle prechádzame od algoritmu k jeho dôkazu, objavujú sa vedľa jednoduchých príkazov typu "zostroj kružnicu nad ...", "veď priamku ...", atď. úvahy o geometrických objektoch. K tomu, aby sme pochopili uvedený dôkaz, potrebujeme poznať Thalesovu vetu a pod.

Aj pri dôkazoch správnosti algoritmov sa budeme stretávať s vlastnosťami objektov, nad ktorými algoritmus pracuje.

Dôkaz správnosti numerického algoritmu bude preto zrejme vychádzať z istých vlastností čísel, dôkaz správnosti programu riadiaceho, či modelujúceho fyzikálne deje sa bude opierať o znalosti príslušných fyzikálnych zákonov.

Príklad:

Teleso je vypustené vo výške h nad zemským povrchom vodorovným smerom rýchlosťou v.
Predpokladáme, že pohyb prebieha vo vzduchprázdnom priestore.
Program dostáva údaje h a v ako vstupné údaje, hodnota g udáva gravitačné zrýchlenie.

Program v jazyku pascal, ktorý má vypisovať súradnice miesta, v ktorom sa nachádza padajúce teleso v jednotlivých sekundách po vypustení, a to až do okamihu, keď dopadne na zem.

```
Program VolnyPad1;
...
x:=0;
t:=0;
y:=h;
repeat
x:=t*v;
y:=h-2*g*t*t;
writeln(x, y);
t:=t+1
until y=0;
...
```

Na prvý pohľad je zrejmé, že program je nesprávny. Obsahuje niekoľko chýb.

Prvá chyba patrí do okruhu, o ktorom sme sa už zmienili. Program totiž nie je v súlade s fyzikálnymi zákonmi. Voľný pád je vyjadrený rovnicou $y = h - g * t * t/2$, nie však $y = h - 2 * g * t * t$. Preto program bude udávať súradnicu y celkom nesprávne.

Program obsahuje aj ďalšie chyby, ktoré s fyzikálnym opisom modelovaného javu nemajú nič spoločné. Chyby, ktoré by sa dali nazvať "čisto programátorske". Umiestnenie príkazu **writeln** spôsobí, že program bude tlačiť aj zápornú hodnotu súradnice y a nebude vypisovať jej počiatočnú hodnotu.

Ďalšou chybou je test **until y=0**, pretože len vo výnimočných prípadoch nadobudne premenná y presne hodnotu 0.

Opravený program

```
Program VolnyPad2;
...
x:=0;
t:=0;
y:=h;
repeat
writeln(x, y);
t:=t+1
x:=t*v;
y:=h-g*t*t/2;
until y<0;
...
```

Pri povrchnom testovaní by niektoré chyby mohli zostať neopravené.

Pokúsme sa predstaviť si, ako by také "ladenie"programu VolnyPad1 mohlo vyzerať.

Najmarkantnejšie sa prejaví test $y = 0$, kvôli ktorému cyklus **repeat - until** pobeží do nekonečna (resp. do preplnenia hodnoty y, vyčerpania časového limitu a pod.). Po zmene podmienky na $y \leq 0$ sa prejaví nevhodné umiestnenie príkazu **writeln** tým, že posledná hodnota y bude záporná. Oprava, pri ktorej presunieme príkaz **writeln** len o jeden riadok vyššie, tento jav odstráni, aj keď samozrejme dáva chybné výsledky.

Hrubá chyba skrývajúca sa za príkazom $y := h - 2 * g * t * t;$ sa však nijako výrazne neprejaví. Túto chybu by odhalilo až porovnanie s experimentálnymi výsledkami alebo človek, ktorý pozná správny tvar príslušnej pohybovej rovnice a dá si prácu s prepočítaním výsledkov.

Pokiaľ sa uspokojíme s povrchným testovaním, ktoré ukáže, že program chodí a dáva "rozumne vyzerajúce výsledky", môžeme prehlásiť za odladený aj celkom chybný program.

Druhy chýb, ktorých sa môžeme pri programovaní dopustiť:

- Syntaktické chyby (napríklad, chýbajúca bodkočiarka, a pod.). Také chyby sú najmenej nebezpečné a odhalí ich už aj prekladač.
- Chyby, ktoré vznikli pri uvažovaní o realite, ktorú program sleduje. Tieto chyby môžu byť neobyčajne vážne a na základe behu programu sú takmer neodhaliteľné. Vymykajú sa tiež zameraniu tejto prednášky.
- "Programátorske chyby", ktoré vzniknú počas popisu modelovanej reality prostriedkami programovacieho jazyka. Práve o ne nám pôjde a budeme študovať spôsoby ako ich eliminovať.

Na dokazovanie správnosti programov už existuje značne prepracovaný formálny aparát.

Nekladieme si za cieľ zoznamovať vás s ním do detailov. Pôjde len o vysvetlenie základných princípov, ktoré umožňujú viesť dôkaz na dostatočnej formálnej úrovni.

Každý programátor sa snaží písať program tak, aby bol správny. Pri tom sa často opiera o úvahy, ktoré nie sú dostatočne prepracované a systematické.
Cesta od pocitu, že by program "mal byť správny"k istote, že tomu tak je, vedie cez spresňovanie a kultivovanie myslenia, o ktoré sa pri písaní programu opiera. Prednáška má k tomu prispieť.

Pri dokazovaní správnosti programov je potrebné mať špecifikované:

a) programovací jazyk, v ktorom budú programy zapísané;
b) špecifikačný jazyk na vyjadrenie vstupnej a výstupnej podmienky a ďalších vzťahov, ktoré platia medzi hodnotami premenných pri vykonávaní príkazov daného programu.

Na splnenie našich cieľov nám postačí predikátový počet 1. rádu ako špecifikačný jazyk a za programovací jazyk v tejto kapitole je zvolený jazyk vývojových diagramov JO.

V súvislosti s používaním predikátového počtu 1. rádu (s rovnosťou) si treba uvedomiť, že problém pravdivosti správne vytvorenej formuly (predikátového počtu 1. rádu) nie je algoritmicky rozhodnuteľný, ale je čiastočne algoritmicky rozhodnuteľný.

Teda existuje algoritmus, ktorý pre danú pravdivú formulu dá o tom, že je pravdivá, odpoveď po konečnom počte krokov, ale ak formula nie je pravdivá, tak buď dá odpoveď po konečnom počte krokov alebo to nezistí.
Tento výsledok má svoj odraz v metódach dokazovania správnosti programov.

K najdôležitejším metódam overovania pravdivosti správne vytvorených formúl predikátového počtu 1. rádu patrí **rezolučná metóda**.

Predikátová logika = predikátový počet = predikátový kalkul = logika kvantifikátorov = logika predikátov = logika funkcionálna

je formálny systém, časť logiky vyšetrujúca spôsoby, ktorými z formúl vznikajú výroky pomocou kvantifikátorov a logických spojok.

Predikátová logika sa zaoberá tak otázkami dokázateľnosti, ako aj pravdivosti. Je rozšírením výrokovej logiky o kvantifikátory a predikátové symboly popisujúce vzťahy (relácie) z univerza.

Okrem bežného predikátového počtu existujú aj predikátové počty vyšších rádov, v ktorých je povolená nielen kvantifikácia objektových premenných, ale aj kvantifikácia predikátov.

Jazyk obsahuje:

- Logické spojky: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- Kvantifikátory: $\exists, \forall$
- Premenné: $x, y, z, \ldots x_1, y_1, ...$
- Konštanty (funkčné symboly s aritou 0): $a, b, c, \ldots$
- Funkčné symboly (horný index k značí aritu funkčného symbolu;$k > 1$): $f^k, g^k, x_1^k, \ldots$
- Predikátové (funkcionálne) symboly (horný index k značí aritu predikátového symbolu; $k > 1$):$P^k, Q^k, R^k \ldots$
- Pomocné symboly $(,), [,], , ,$

Vyjadrite formálne:

1. Nikto, kto nie je zapracovaný (P), nepracuje samostatne (S).
2. Nie každý talentovaný (T) spisovateľ (Sp) je slávny (Sl).
3. Len zamestnanci (Z) používajú výťah (V).
4. Nie každý človek (C), ktorý veľa hovorí (M), nemá čo povedať (R).
5. Niekto je spokojný (Sn) a niekto nie je spokojný.
6. Niektorí šikovní ľudia (Ch) sú leniví (L).
7. Všetci zamestnanci (Z) používajú výťah (V).

Pomôcka: Po všeobecnom kvantifikátore nasleduje formula v tvare implikácie, po existenčnom ...

Vyjadrite formálne:

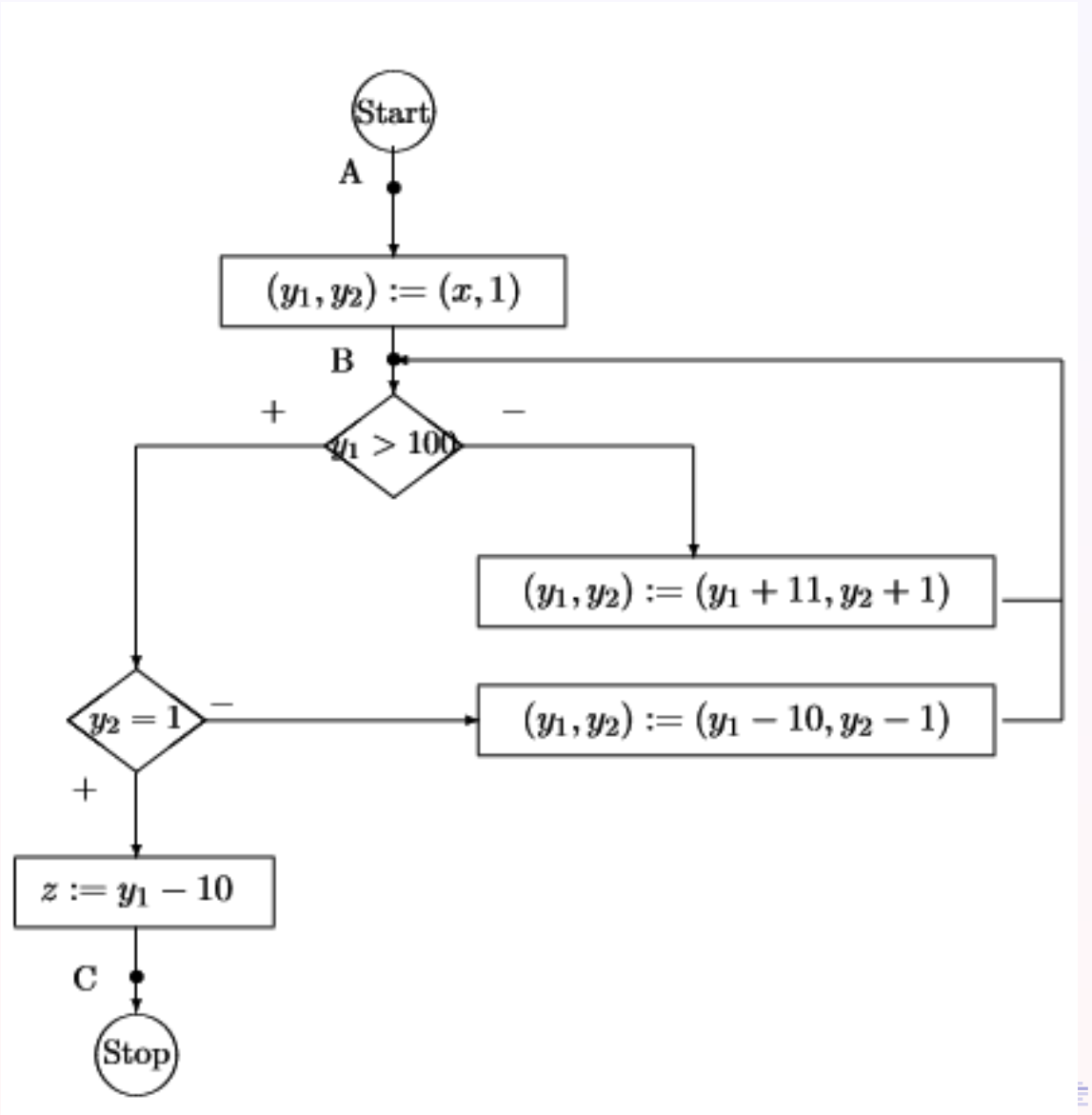
1. Nikto, kto nie je zapracovaný (P), nepracuje samostatne (S).
2. Nie každý talentovaný (T) spisovateľ (Sp) je slávny (Sl).
3. Len zamestnanci (Z) používajú výťah (V).
4. Nie každý človek (C), ktorý veľa hovorí (M), nemá čo povedať (R).
5. Niekto je spokojný (Sn) a niekto nie je spokojný.
6. Niektorí šikovní ľudia (Ch) sú leniví (L).
7. Všetci zamestnanci (Z) používajú výťah (V).

Pomôcka: Po všeobecnom kvantifikátore nasleduje formula v tvare implikácie, po existenčnom ...

K prvému zoznámeniu sa s Floydovou metódou nám poslúžil program v **jazyku vývojových diagramov** pre výpočet **McCarthyho funkcie 91**.

Tvrdíme, že tento program počíta funkciu $f(x)$ nad oborom celých čísel definovanú predpisom:

$$z = f(x) = \begin{cases} x - 10, & \text{pre } x > 100, \\ 91, & \text{inak.} \end{cases} \tag{1}$$


Start
A
$(y_1, y_2) := (x, 1)$
B
+
$y_1 > 100$
−
$(y_1, y_2) := (y_1 + 11, y_2 + 1)$
$y_2 = 1$
−
$(y_1, y_2) := (y_1 - 10, y_2 - 1)$
+
$z := y_1 - 10$
C
Stop

Dôkaz McCartyho funkcie

1. Predpokladajme, že $x > 100$.
   Výpočet prebieha po ceste bez cyklov $AB++C$, a teda skončí a platí $z = x - 10$.
   Pomocou znamienok $+, -$ budeme vyjadrovať smer postupu pri vetvení (v podmienkových príkazoch). Smer postupu pri splnenej podmienke je označený $+$, pri nesplnenej $-$.

2. Obtiažny je len prípad $x \leq 100$. Dôkaz pre tento prípad urobíme v dvoch krokoch.
   2.I. Najprv dokážeme, ak program skončí, vydá výsledok $z = 91$.
   Presvedčme sa najprv, že kedykoľvek výpočet prechádza bodom B, vždy je v ňom splnená podmienka:

$$(y_1 \leq 101 \wedge y_2 \geq 1) \vee (y_1 \leq 111 \wedge y_2 > 1). \qquad (2)$$

Dôkaz McCartyho funkcie

Platnosť (2) overíme indukciou:

- pri prvom prechode bodom B je $y_1 \leq 100 \wedge y_2 = 1$, t.j. uvedená podmienka platí;
- ukážeme, že ak je podmienka splnená pri (i)-tom prechode, bude splnená aj pri nasledujúcom (i+1)-vom prechode
  a) Ak je splnená podmienka a navyše $y_1 \leq 100$, vykoná sa príkaz $(y_1, y_2) := (y_1 + 11, y_2 + 1))$, takže po návrate do bodu B platí, že $y_1 \leq 111 \wedge y_2 > 1$, t.j. je splnený druhý člen uvedenej disjunkcie.
  b) Ak je splnená uvedená podmienka, t.j. určite platí $y_1 \leq 111$ a navyše $y_1 \geq 101$, potom, ak sa má výpočet vrátiť do bodu B, musí byť $y_2 > 1$. Je vykonaný príkaz $(y_1, y_2) := (y_1 - 10, y_2 - 1))$, t.j. po návrate do bodu B je splnený prvý člen uvedenej disjunkcie: $y_1 \leq 101 \wedge y_2 \geq 1$.

Tým sme overili platnosť vzťahu (2).

Dôkaz McCartyho funkcie

Ak niektorý výpočet skončí, vykoná sa posledný úsek programu z bodu B do bodu C. Teda prejde po ceste $B++C$, čo znamená, že $y_1 > 100$ a $y_2 = 1$, a preto podľa (2) pri poslednom prechode bodom B je $y_1 = 101 \wedge y_2 = 1$.
Po vykonaní príkazu $z := y_1 - 10$ je $z = 91$.

2. II. Zostáva nám dokázať, že pre každé $x \leq 100$ program skončí.
Uvažujme výraz $k = -2 * y_1 + 21 * y_2 + 202$.
Podľa prvej časti dôkazu je pri každom prechode bodom B určite $y_1 \leq 111 \wedge y_2 \geq 1$. Preto je hodnota k vždy kladná.
Ľahko možno overiť, že aj vykonaním príkazu
$(y_1, y_2) := (y_1 + 11, y_2 + 1))$ aj vykonaním príkazu
$(y_1, y_2) := (y_1 - 10, y_2 - 1))$, sa hodnota k zmenší.

Pri každom ďalšom prechode bodom B má k hodnotu menšiu než pri predchádzajúcom prechode. Teda hodnoty výrazu k v okamihu prechodu bodom B tvoria klesajúcu postupnosť kladných čísel. taká postupnosť je však konečná.
Teda výpočet nemôže uviaznuť v cykle začínajúcom aj končiacom v bode B a po konečnom počte prechodov bodov B skončí v bode C.
Tým je dôkaz správnosti uvedeného programu ukončený. □

Úvahy, ktoré viedli k vytvoreniu podmienky

$$(y_1 \leq 101 \wedge y_2 \geq 1) \vee (y_1 \leq 111 \wedge y_2 > 1)$$

a výrazu $-2 * y_1 + 21 * y_2 + 202$ sú kvôli stručnosti vynechané a pre overenie korektnosti programu nie sú podstané (podobne ako v dôkazoch typu: "nech $\epsilon > 0$ je ľubovoľné, zvoľme $\delta = 125 * \epsilon/7 \ldots$".).