Kybernetika

Eduard Kostolanský

Jazyk SNOBOL I a jeho realizácia

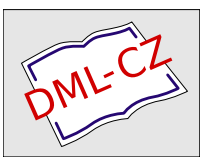
Kybernetika, Vol. 6 (1970), No. 4, (280)--290

Persistent URL: http://dml.cz/dmlcz/125313

Terms of use:

© Institute of Information Theory and Automation AS CR, 1970

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



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

KYBERNETIKA ČÍSLO 4, ROČNÍK 6/1970

Jazyk SNOBOL I a jeho realizácia

EDUARD KOSTOLANSKÝ

Článok neformálnym spôsobom popisuje štruktúru programovacieho jazyka SNOBOL I. Ďalej podáva jeden spôsob jeho realizácie, ktorú tvoria dve etapy, etapa translácie a etapa interpretácie.

ÚVOD

Možno predpokladať, že v krátkej budúcnosti sa spôsob spojenia človeka s počítačom bude uskutočňovať vo forme, ktorá viacmenej bude pripomínať ľudský dialóg. Vo všeobecnosti, ak sa má uskutočňovať nejaký užitočný rozhovor, zúčastnené stránky, diskutujúci musia mať spoločný jazyk. V našom prípade to znamená, že počítač musíme naučiť našej reči, ktorá sa žiaľbohu bude rozlišovať od bežnej prirodzenej reči svojou presnosťou a jednoznačnosťou. V tomto konštatovaní netreba vidieť žiadnu nevýhodu pre nás ľudí. Je totiž nie veľmi príjemné predstaviť si dobu, v ktorej počítače budú schopné akceptovať dialógy v bežnej hovorovej reči a podľa toho aj konať.

Nám však zatiaľ ide o skutočnosť, že k realizácii spomenutého dialógu medzi človekom a počítačom je nutné vytvárať algoritmy, v ktorých základnú úlohu budú mať operácie nad reťazcami symbolov — textami.

K požiadavke konštrukcie algoritmov, v ktorých ako základný údaj vystupuje reťazec symbolov — text, sa dostane aj pri použití počítačov na iné etapy pri riešení ľubovoľného problému. Riešenie problému vo všeobecnosti pozostáva z následujúcich etáp:

1. Formulácia problému (matematická).
2. Hľadanie algoritmu, alebo vhodnej metódy pre riešenie.
3. Zápis algoritmu vo forme akceptovateľnej počítačom — programovanie.
4. a) Kompilácia programu;

b) Realizácia programu počítačom — výpočty úloh s konkrétnymi parametrami, z ktorých počítač uskutočňuje zatiaľ iba poslednú etapu. Ak by sme napríklad chceli prenechať aj programovanie na úrovni niektorého z používaných programovacích jazykov (ALGOL, FORTRAN, COBOL, atď.) počítaču, tak by sme boli nútení vytvoriť algoritmy, výsledkami ktorých by bolo transformovanie výpočtových procedúr, popísaných na úrovni bežných formulačných prostriedkov (prirodzený text, matematické, inžinierske a pod. vyjadrovacie prostriedky) do zápisu v niektorom z uvedených jazykov. Teda v podstate by sa jednalo o algoritmy, ktoré nejakým

spôsobom transformujú vstupný text do istého výstupného textu. Je možné usudzovať, že požiadavke konštruovať algoritmy spomenutého typu sa nevyhneme pri realizácii 2. a 1. etapy riešenia problému.

V krátkosti by sme proces realizujúci 1. a 2. etapu, mohli charakterizovať ako:

1. etapa: Stavanie otázok týkajúcich sa vzájomných súvislostí medzi presne definovanými pojmami.

2. etapa: Nachádzanie konštruktívnych postupov, ktoré vedú k zodpovedaniu otázok postavených v 1. etape.

Pravda, nie je cieľom tejto práce bližšie analyzovať problematiku, ktorá sa skrýva pod názvami „formulovanie problému" a „spôsob hľadania konštruktívnych postupov".

Chcem však poukázať len na to, že riešenie týchto problémov predstavuje časť simulovania ľudského poznávacieho procesu a zdá sa nám, že algoritmy, ktoré majú reprezentovať, pokiaľ je to možné, ľudský poznávací proces, budú mať ako základný údaj text.

S cieľom vypracovania vhodného programovacieho systému pre popis uvedených algoritmov sme pristúpili k definovaniu syntaxu a sémantiky jazyka, vhodného pre popis algoritmu úloh uvedeného typu.

1. JAZYK SNOBOL I – SYNTAX A SÉMANTIKA

Pri definícii jazyka SNOBOL I sme vychádzali z predpokladu, že k algoritmizácii problémov, pre ktoré je určený jazyk SNOBOL I., treba uvažovať s nasledujúcimi operáciami:

a) formovanie reťazcov;
b) analýza reťazcov s cieľom určenia istých vlastností analyzovaného reťazca;
c) zmena reťazcov na základe predchádzajúcej analýzy.

(Obdobné operácie tvoria jadro jazyka SNOBOL, ktorého prvá verzia bola skrátene popísaná v časopise JACM, January 1965. Tiež je známe, že bolo urobených už niekoľko verzií tohoto jazyka, posledná je rozpracovaná u fy Bell Telephone Co. Autorovi sa však nepodarilo získať bližšie informácie o tomto systéme.)

Presná definícia syntaxu a sémantiky jazyka SNOBOL I. je publikovaná v [3]. Cieľom tejto práce je popisovým spôsobom objasniť štruktúru jazyka SNOBOL I. a naznačiť jeden zo spôsobov jeho realizácie na počítači.

Základný pojem v SNOBOL I., používaný pre popis uvedených algoritmických procesov je *reťazcový výraz*. Jeho zložkami sú *reťazce – konštanty*, *reťazcové premenné* a *obmedzovače*. Reťazce sú tvorené z prvkov abecedy (abeceda môže byť vhodne rozšírená alebo zúžená).

Ak sa napríklad obmedzíme na abecedu tvorenú písmenami, číslicami, aritmetickými operátormi ako +, – atď., bežnými znakmi interpunkcie, potom reťazce majú tvar napr.

program , meno 1 , podmienka = a pod.

V súvislosti s reťazcami hovoríme o *reťazcových hodnotách*. Reťazce majú hodnotu samých seba. V ďalšom reťazcové hodnoty budeme uvádzať v dvojici zátvoriek ", ".

K označeniu reťazcovej hodnoty slúži reťazcová premenná, ktorú tvorí reťazec, pred ktorým predchádza symbol α. Operácia *zreťazenia* je reprezentovaná operátorom & a vystupuje v *reťazcovom výraze*, ktorý slúži k získaniu novej reťazcovej hodnoty. Ak napr. reťazcové premenné α meno, α vek mali hodnoty „Slávik" a „33", potom vyhodnotením výrazu α meno & α vek & schopný sa získa nová hodnota „Slávik 33 schopný".

Priradenie novej hodnoty premennej možno uskutočniť viacerými spôsobmi. Najjednoduchší z nich je cez *priraďovací príkaz*, ktorý má tvar

premenná := *reťazcový výraz*

Napr. ak premenné α a, α a1 mali hodnotu „podmienka" a „splnená", potom príkazom

α a2 := α a & nie je & α a1

sa premennej α a2 priradí hodnota, „podmienka nie je splnená".

Iný spôsob priradenia hodnôt premenným je pomocou *príkazu priradenia podľa vzoru*. Pri tomto príkaze sa premennej priradí tá časť reťazca, ktorá sa nachádza medzi jeho časťami, udanými vzorom.

Zápis tohoto príkazu voľne možno nasledovne označiť

analyzovaná premenná ⌋ *vzor* * *premenná* * *vzor* .

Priradenie v tomto príkaze sa vykoná len vtedy, ak v analyzovanej premennej sa nachádzajú reťazce udané vzormi. Majme napr. premennú α krok, ktorá má hodnotu

„Vypočítať $x = 2a + b$ koniec!"

Príkazom priradenia podľa vzoru

α krok ⌋ vypočítať * α algoritmus * koniec

sa premennej α algoritmus priradí hodnota „$x = 2a + b$". V prípade, že v analyzovanom reťazci vystupuje viackrát vzor, hľadá sa len jeho prvý výskyt. Teda účinok uvedeného príkazu priradenia podľa vzoru zostáva ten istý, ak premenná α krok mala hodnotu

„Vypočítať $x = 2a + b$ koniec! Vypočítať $\exp(2a + b)$ koniec!"

V príkaze priradenia podľa vzoru môže vystupovať aj viacej požiadavok priradenia. Pri každom ďalšom analyzovaní udaného reťazca sa za jeho začiatok berie prvý symbol, ktorý nasleduje za vzorom z predchádzajúceho priradenia. Napr. ak α a má hodnotu „ak $a = b$, treba vypočítať hodnotu funkcie $y = \exp(2x + 3)$!", potom príkazom priradenia podľa vzoru

α a ⌋ treba vypočítať * α a1 * y ⁄ = * α a2 * !

sa premennej α a1 priradí hodnota „hodnotu funkcie" a α a2 bude mať hodnotu „exp $(2x + 3)$". Hodnota α a sa nemení. Teda príkazom priradenia podľa vzoru sa časti reťazca, ktorý je hodnotou niektorej premennej, priraďujú premenným ako ich nové hodnoty.

Širší účinok ako príkaz priradenia podľa vzoru má *zámena*. Okrem priradenia hodnôt premenným, tak ako je to popísané v príkaze priradenia podľa vzoru nahrádza v analyzovanom reťazci tú jeho časť (alebo časti), ktorú tvoria úseky označené ako „vzor, premenná, vzor". Táto časť sa nahradí reťazcovou hodnotou vpravo od ≐.

Tak napríklad ak α a má hodnotu

„Sme z hliny a z blata"

potom príkaz zámeny

α a ⌋ z * α b * a ≐ páni

má tento účinok : 1. premennej α b priradí hodnotu „hliny" a 2. premenná α a bude mať novú hodnotu „Sme páni z blata".

Príkaz zámeny je pokračovaním ľubovoľného príkazu priradenia podľa vzoru. Teda možno ním nahradiť viacej častí analyzovaného reťazca tou istou hodnotou.

Tak napríklad, ak α a mala hodnotu

„Nie je všetko pravda, ale je pravda"

príkaz zmeny α a ⌋ je * α b * pravda / je * α c * pravda ≐ tak, má nasledovný účinok. Premennej α b sa priradí hodnota „všetko", hodnotou premennej α c bude prázdny reťazec a reťazec „Nie tak, ale tak" je novou hodnotou premennej α a.

Ľubovoľný príkaz môže byť opatrený *náveštím*, ktoré je reprezentované reťazcom z prvkov abecedy. Postupnosť *príkazov programu* tvorí *program* v SNOBOL I. Tieto príkazy sa vykonávajú postupne za sebou, alebo môže byť určené, ktorý príkaz sa bude vykonávať ako ďalší. K tomuto účelu slúži *príkaz skoku*, ktorý môže prakticky vystupovať ako samostatný príkaz programu, alebo spolu s príkazmi priradenia a zámeny môže tvoriť jeden príkaz programu.

Nepodmienený príkaz skoku, ktorý má tvar ⊦ *náveštie*, má za následok, že ako ďalší sa bude vykonávať príkaz, opatrený s príslušným náveštím. Tak napr. príkaz programu

α a ⌋ je * α b * pravda / je * α c * pravda ≐ tak ⊦ PRX ;

okrem účinku, ktorý je popísaný vyššie má za následok, že ako ďalší sa bude vykonávať príkaz opatrený s náveštím PRX.

Podmienený príkaz skoku má syntax **T** *náveštie*, alebo **F** *náveštie*. Účinok podmieneného skoku závisí

a) buď od zbývajúcej časti príkazu programu, v ktorom sa tento podmienený príkaz skoku vyskytuje,

b) alebo od *označenia logickej hodnoty*, ktoré môže byť súčasťou príkazu skoku.

V prípade a) ide o nasledovnú závislosť. Zbývajúcou časťou príkazu programu, tj. príkazom priradenia podľa vzoru, alebo zámenou sa vytvorí *logická hodnota*, označená **TRUE**, ak tieto príkazy boli úspešné, teda ak sa vykonalo priradenie podľa vzoru, alebo zámena. V opačnom prípade sa vytvorí logická hodnota **FALSE**. Pri hodnote **TRUE** podmienený príkaz skoku **T** *náveštie* spôsobí, že ako ďalší sa bude vykonávať príkaz opatrený príslušným náveštím. Pri hodnote **FALSE** sa príkaz **T** *náveštie* ignoruje, tj. bude sa vykonávať v poradí ďalší príkaz.

Obdobné pravidlo platí aj pre podmienený príkaz skoku **F** *náveštie*: Pri neúspešnej zbývajúcej časti príkazu programu **F** *náveštie* má účinok nepodmieneného príkazu skoku, v opačnom prípade sa ignoruje. Uveďme príklady. Predpokladajme, že α x má hodnotu

„Nájsť reálny koreň polynómu".

Príkaz α x ⌋ Nájsť ∗ α *y* ∗ polynómu **T** PR2;
má nasledujúci účinok. Premennej α y sa priradí hodnota „reálny koreň" a ako ďalší sa vykonáva príkaz s náveštím PR2.

Pri tej istej hodnote α x príkaz

α x ⌋ komplexný ∗ α y ∗ polynómu **F** PR3 ;

má účinok: Premennej α y sa nepriradí žiadna hodnota a ako ďalší sa vykonáva príkaz s náveštím PR3.

Z podmieneného a nepodmieneného príkazu skoku možno tvoriť zložený príkaz skoku, ktorý má syntax **T** (alebo **F**) *náveštie* ⊦ *náveštie*. Tento realizuje požiadavku, aby pri prípadnom ignorovaní podmieneného príkazu skoku sa vykonával v poradí ďalší príkaz, ale ľubovoľný príkaz programu, opatrený príslušným náveštím.

Napríklad, ak α x mala hodnotu ako doteraz, potom príkaz

α x ⌋ Nájsť ∗ α y ∗ polynómu **F** PR1 ⊦ PR2 ;

má nasledovný účinok. Premennej α y sa priradí hodnota „reálny koreň" a ako ďalší sa vykonáva príkaz s náveštím PR2, pretože podmienený príkaz **F** PR1 sa ignoruje.

Pri uvedenej hodnote α x príkaz

α x ⌋ Nájsť ∗ α y ∗ **T** PR4 ⊦ PR5 ;

premennej α y priradí hodnotu „reálny koreň polynómu" a ako ďalší sa vykonáva príkaz s náveštím PR4.

Logická hodnota, ktorá sa získa pri niektorom príkaze programu môže byť v spojení s podmieneným, alebo zloženým príkazom skoku použitá k vetveniu programu na ľubovoľnom mieste. Jej uchovanie sa uskutočňuje pomocou označenia logickej hodnoty, ktorá môže vystupovať samostatne, alebo v spojení s nepodmieneným príkazom skoku. Syntax označenia logickej hodnoty je Γ *premenná*. Napríklad: Nech α z má hodnotu „určiť tretí člen". Potom príkazom

α z ⌋ určiť ∗ α v ∗ člen ≐ Γ α b ⊦ PR ;

sa priradí α v hodnota „tretí", nová hodnota α z je prázdny reťazec, premennej α b sa priradí logická hodnota **TRUE** a pokračuje sa na príkaze s náveštím PR.

Ak premennej v označení logickej hodnoty sa priradila už logická hodnota, potom táto môže byť použitá k riadeniu účinku podmieneného, alebo zloženého príkazu skoku na ľubovoľnom mieste programu.

Uvažujme napríklad nasledujúce príkazy za predpokladu, že α z má hodnotu „Politika je pekná vec".

```
α z ⌋  Politika ∗ α v ∗ vec ⊢ α b ;
α z ⌋  rozumná ≐ hlúpa  ⌈ α b  T  PR1  ⊢ PR2 ;
```

Prvým príkazom sa α v priradí hodnota „je pekná" a premennej α b logická hodnota **TRUE**. Hodnota α z sa nemení. Druhý príkaz zisťuje, či α z obsahuje časť „rozumná", ktorá by sa nahradila hodnotou „hlúpa". Pretože tomu tak nie je, hodnota α z sa nemení a ako ďalší sa bude vykonávať príkaz s náveštím PR1, pretože α b má logickú hodnotu **TRUE**.

Uveďme ešte niektoré poznámky k premenným a náveštiam. Z uvedeného popisu vyplýva, že ľubovoľná premenná môže mať ako logickú, tak aj reťazcovú hodnotu. Programátor musí mať na zreteli túto možnosť, hlavne ak tej istej premennej sa striedavo priraďujú kvalitou rôzne hodnoty.

V uvedených príkladoch vždy vystupovalo priame náveštie, tj. nejaký reťazec. Ako náveštie možno používať aj premennú, ktorej sa predtým priradil nejaký reťazec, ktorý je náveštím niektorého príkazu programu, ktorý treba spracovať. Potom hovoríme o *nepriamom náveští.*

Príklad programu v SNOBOL I

Nech $\alpha_1, \alpha_2, \ldots, \alpha_n$ sú symboly s klesajúcou prioritou (teda α_1 má najväčšiu a α_n najmenšiu prioritu). Reťazec $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$, ktorý je tvorený z týchto symbolov, treba transformovať do tvaru $\alpha_{j_1}, \ldots, \alpha_{j_k}$ v ktorom priorita α_{j_i} (pre $1 \leqq i < k$) je väčšia ako priorita $\alpha_{j_{i+1}}$.

Program pre realizáciu je nasledovný:

```
        α P  :=  α_i1, α_i2, ..., α_ik ;
        α N  :=  ;
        α M  :=  α_1, α_2, ..., α_n ;
PR1     α M⌋  ∗ α D ↖ 1 ∗,  ≐ F   END ;
PR2     α P⌋     α D          ≐ F   PR1 ;
PR3     α N⌋     α N := αN & αD   ⊢ PR2 ;
END
```

V tejto kapitole uvedieme základné vlastnosti navrhovaného systému pre spracovanie programov v SNOBOL I. Jeho hlavnými zložkami sú *prekladací program* (*translátor*) a *interpretačný program* (*interpretátor*). Systém bude napísaný pre počítač GIER. Pri jeho navrhovaní sa však neprizeralo na zvláštnosti tohoto počítača, takže navrhovaný spôsob spracovania programov v SNOBOL I. môže byť použitý pri realizovaní jazyka SNOBOL I. na ľubovoľnom počítači. V súhlase s prácou translátora a interpretátora hovoríme o etape prekladania a o etape interpretovania programu v SNOBOL I. Výsledkom translácie nie je program v strojovom kóde počítača, ale program na úrovni makroinštrukcií. Každá makroinštrukcia udáva, aký príkaz sa ňou realizuje a ktoré operandy sú potrebné k realizácii príkazu. Po preložení vstupného programu do makroinštrukcií sa spracováva program makroinštrukcií interpretačným spôsobom. Popíšme teraz bližšie činnosť na jednotlivých etapách.

2.1 Translácia

Proces translácie sa realizuje v troch prechodoch. Prvý prechod transformuje jednotlivé symboly do číselného tvaru, kontroluje prípustnosť jednotlivých symbolov, a pokiaľ je to možné, odhaluje syntaktické chyby u zložených symbolov, ktoré vznikli pri písaní, alebo dierovaní programu. Teda výsledkom prvého prechodu je zakódovanie vstupného textu programu do tvaru celých čísiel.

Počas druhého prechodu sa vytvárajú nasledovné informačné tabuľky. Tabuľka konštánt a premenných, ktorej riadky obsahujú lokalizované číslo v rámci tabuľky, údaj o druhu veličiny (konštanta, alebo premenná), udanie dĺžky a umiestnenie v pamäti. Posledné dva údaje sa pri premenných dopĺňajú na etape interpretácie. Konštanty sa ukladajú postupne za sebou v určenom segmente pamäti. Tabuľka náveští obsahuje lokalizačný údaj v rámci tabuľky, lokalizáciu príkazu, ku ktorému patrí a vlastné náveštia. Potreba uchovania vlastného náveštia vyplýva z možností nepriameho náveštia.

Okrem tvorenia uvedených tabuliek sa v rámci druhého prechodu konštanty, premenné a náveštia nahrazujú lokalizačným údajom v rámci tej ktorej tabuľky.

Tretí prechod uskutočňuje syntaktickú kontrolu a vytvára makroinštrukcie. Nerekurzívna definícia syntaxu SNOBOL I. dovoľuje vytvárať syntaktickú kontrolu a prekladanie do makroinštrukcií pomocou podprogramov pre jednotlivé typy príkazov. Algoritmy syntaktickej kontroly identifikujú syntaktické chyby a dávajú informácie o mieste a druhu chyby. Pokiaľ je to možné, tiež „opravujú" chyby a to tak, aby opravená časť v kontexte dávala niektorú správnu syntaktickú jednotku jazyka.

Cieľom takéhoto spôsobu syntaktickej kontroly je pri jednom zavedení programu nájsť čo najviac syntaktických chýb a vždy ukončiť transláciu celého programu.

Makroinštrukcie, ktoré sa postupne tvoria počas tretieho prechodu k jednotlivým príkazom, sa skladajú z riadiacej časti a časti informačnej.

Riadiaca časť obsahuje odvolávku na časť interpretátora, ktorá realizuje danú makroinštrukciu. Informačná časť obsahuje údaje o operandoch, ako napr. počet operandov príkazu, resp. jeho časti, ich rozmiestnenie v pamäti a pod.

Základná požiadavka, kladená na formu makroinštrukcií je, aby poradie údajov v informačnej časti súhlasilo s poradím ich spracovania na etape interpretovania.

Ako príklad tvaru makroinštrukcie uveďme makroinštrukciu príkazu jednoduchého priradenia. Tá je nasledovná $\rightarrow R, a, b1, \ldots, bk$, kde $\rightarrow R$ definuje prechod do príslušnej časti interpretačného programu, a je lokalizácia premennej, ktorej sa priraďuje hodnota, v tabuľke premenných.

$b1, \ldots, bk$ sú informácie o umiestnení operandov v tabuľke; jedná sa o operandy reťazcového výrazu, ktorý vystupuje v príkaze. Obdobný tvar majú aj makroinštrukcie iných príkazov.

Proces translácie formálnejšie možno popísať nasledovne. V rámci translácie sa realizujú transformácie $T_c(L_v) = L_c$, $T_i(L_c) = L_i$, kde T_c realizuje transformovanie vstupného programu v L_v do celočíselnej formy v L_c. Výsledkom T_i je transformovanie do vnútorného jazyka L_i, vhodného pre interpretáciu. Na L_i je kladená požiadavka, aby dĺžka ľubovoľného príkazu programu bola menšia, alebo rovná dĺžke tohoto príkazu uvažovanom v L_c.

2.2 Interpretácia

Program preložený do makroinštrukcií sa realizuje na počítači pomocou interpretačného programu. Interpretačný program tvoria podprogramy, ktoré sú určené k realizácii jednotlivých makroinštrukcií.

Z hľadiska realizácie jednotlivých operácií definovaných v SNOBOL I. na počítači, tieto predstavujú uchovávanie a premiestňovanie segmentov pamäti, ktoré obsahujú informáciu (reťazce symbolov) prístupnú cez niektorý názov, premennú. Segmenty, označené tým istým názvom, tj. hodnoty, ktoré sa priraďujú tej istej premennej, môžu mať prirodzene rôznu dĺžku. Pri prvom pamätaní sa pamätajú tieto segmenty kontinuálne za sebou. Pri ďalších priraďovaniach môže nastať, situácia, že novopriraďovaná hodnota má menšiu, resp. väčšiu dĺžku ako pôvodná.

Teda v prvom prípade vznikne medzera medzi segmentami a v druhom prípade by tento segment prekryl časť nasledujúceho, prípadne niekoľko ďalších segmentov, určených pre iné premenné.

Tento problém u navrhovaného translátora sa rieši nasledovne.

Vytvára sa zoznam voľných úsekov pracovného poľa pamäti. Prvky zoznamu obsahujú lokalizovanie a dĺžku týchto úsekov. Voľné úseky vznikajú v oboch prípadoch i keď je novopriradená hodnota premennej reprezentovaná kratším segmentom než bol pôvodný, uchovávajúci predchádzajúcu hodnotu premennej, alebo dlhším. V prvom prípade sa zoznam rozšíri o údaje úseku, ktorý sa vytvorí zbývajú-

cou časťou pôvodného segmentu premennej. V druhom prípade sa do zoznamu voľných úsekov zapíšu údaje o celom segmente pôvodnej hodnoty premennej.

Nová hodnota premennej sa uloží od začiatku voľnej časti pracovného poľa. (Údaj o začiatku voľnej časti pracovného poľa sa stále uchováva.) Toto má za následok aj zmenu lokalizačných údajov v tabuľke premenných.

Vyššie popísaný algoritmus vkladania nových hodnôt v pamäti je súčasťou interpretačných podprogramov, ktoré realizujú makroinštrukcie príkazov priradenia.

Popíšme ešte v krátkosti hlavné činnosti interpretačných programov k jednotlivým makroinštrukciám.

Podprogram na realizovanie jednoduchého priradenia

Zistí, či sa príkaz interpretuje prvý krát. Ak áno, priraďovaná hodnota sa pamätá od začiatku voľného úseku pracovného poľa. Informácia o umiestnení a dĺžke tejto hodnoty sa zapíše do tabuľky premenných. V opačnom prípade, ak príkaz sa neinterpretuje prvý krát, sa umiestnenie priraďovanej hodnoty vykoná algoritmom, popísaným na začiatku tejto kapitoly.

Podprogram na realizovanie priradenia podľa vzoru

Jeho činnosť je obsiahnutá v nasledujúcich bodoch:

rozhodne, ktorý reťazec treba analyzovať;

ku každému vzoru určí začiatok tej časti analyzovaného reťazca, v ktorej treba hľadať obraz k danému vzoru;

hľadá obraz k príslušnému vzoru podľa informácií o operandoch, ktoré tvoria vzor;

uchováva údaje (dĺžku, umiestnenie v pamäti) o tých častiach analyzovaného reťazca, ktoré môžu byť novými hodnotami premenných, ak príkaz priradenia podľa vzoru bude úspešný;

ak k niektorému vzoru nie je nájdený obraz, v ďalšej analýze sa nepokračuje a účinok celého programu sa redukuje na získanie logickej hodnoty **FALSE**;

pri nájdení obrazov ku všetkým vzorom sa získa logická hodnota **TRUE** a priradia sa určené hodnoty k príslušným premenným.

Podprogram pre príkaz zámeny

Podprogram je v spojení s predchádzajúcim podprogramom pre priradenie podľa vzoru. Ak tento bol úspešný, tj. sa získa logická hodnota **TRUE**, potom podprogram pre zámenu uskutočňuje substitúciu príslušných úsekov analyzovaného reťazca reťazcom, ktorý je hodnotou reťazcového výrazu, vystupujúceho v zámene vpravo od ohraničiteľa =.

Podstata činnosti tohoto podprogramu pri jeho volaní spočíva v určení poradia spracovania príkazov, prípadne uchovania logickej hodnoty. Ak prichádza do úvahy iba uchovanie logickej hodnoty, potom sa nemení postupné prirodzené spracovanie príkazov. V iných prípadoch treba určiť pomocou tabuľky náveští, ktorý príkaz sa bude realizovať ako ďalší. Ako už bolo spomenuté, tabuľka náveští obsahuje náveštia a informáciu o umiestnení príkazov v pamäti, ku ktorým tieto náveštia patria. Pri nepriamom náveští podprogram hľadá zhodné náveštie v tabuľke náveští k náveštiu, ktoré je hodnotou premennej vystupujúcej v príkaze skoku. Možnosť nepriameho náveštia vyžaduje v tabuľke náveští uchovať všetky náveštia vyskytujúce sa v programe.

Realizácia vstupu a výstupu

Vstup je realizovaný príkazom *input* (*prem* 1, *prem* 2, ..., *prem* n), kde *prem i* sú reťazcové premenné. Reťazec, ktorý má byť novou hodnotou premennej a sa tejto premennej priraďuje prostredníctvom príkazu vstupu, končí symbolom *end of string*. Teda na vstupe sú jednotlivé reťazce, ktoré sa majú priradiť, ukončené *end of string*.

Vlastné priradenie sa uskutočňuje spôsobom popísaným pri podprograme pre jednoduché priradenie.

Výstup sa realizuje príkazom výstupu *write* (*reťazcová premenná*). Jeho účinok spočíva v odovzdaní hodnoty reťazcovej premennej niektorému výstupnému médiu.

Činnosť na etape interpretácie je koordinovaná riadiacim programom. Jeho súčasťou je aj podprogram na organizáciu pamäti. Tento podprogram môže byť volaný počas interpretácie ľubovoľnej makroinštrukcie, keď sa zaplnilo pracovné pole pamäti a je požiadavka pamätať do voľného úseku pracovného poľa. Predpokladá sa, že medzi hodnotami premenných v pracovnom poli sú voľné úseky. Účinok podprogramu pre organizovanie pamäti je spojité pamätanie hodnôt premenných a získanie voľného úseku pracovného poľa. Pravda, takto vykonaná organizácia pamäti si vyžaduje aj zmenu lokalizačných údajov v tabuľke premenných.

Detailnejší popis realizácie systému, zvlášť s prihliadnutím na rozdelenie pamäti, spolu s praktickými skúsenosťami dáme po jeho konkrétnej realizácii a overení.

(Došlo dňa 9. júna 1969.)

LITERATÚRA

[1] D. G. Bobrow, B. Raphael: A Comparison of List — Processing Computer Languages, Including a Detailed Comparison of COMIT, IPL — V, Lisp 1.5 and SLIP. Commun. ACM (April 1964).

[2] D. Farber, R. E. Griswold, Polonsky: SNOBOL, A String Manipulation Language. Journal of ACM (January 1964).

[3] E. Kostolanský: Definícia syntaxe a sémantiky jazyka SNOBOL I. Kybernetika *3* (1967), 3, 253—268.
[4] W. L. van der Poel: The software crisis: some thoughts and outlooks. Processing of IF IP Congres 1968.
[5] M. Engeli: Achievements and problems in formula manipulation. Processing of IF IP Congres 1968.

SUMMARY

The Language SNOBOL I and its Implementation

EDUARD KOSTOLANSKÝ

The paper describes in an informal way the syntax and semantics of the programming language SNOBOL I and one of the modes of its implementation.

The language belongs to the group of symbol – manipulation languages. In SNOBOL I there is actually a sole type of data – the string. There is no description of variables. The program in SNOBOL I is formed by a sequence of statements which can be preceded by a label. Actually, there are two types of statements: assignment statements, jump statement.

The properties of the language have an impact upon its implementation. The suggested way of implementation endeavours, as far as possible, to be general and machine-independent. The proper SNOBOL I system consists of a compiler and an interpreter. The result of the stage of compiling is the transformation of the input program into an inward form of macroinstructions suitable for interpretation. Each macroinstruction is associated with an interpreting subroutine. An essential property of the system is a dynamic storage allocation which may be activated by an arbitrary interpretation subroutine.

RNDr. Eduard Kostolanský, Ústav technickej kybernetiky SAV, Dúbravská cesta, Bratislava.