RECENZE/KYBERNETIKA ČÍSLO 3, ROČNÍK 3/1967

# Definícia syntaxu a sémantiky jazyka SNOBOL I

EDUARD KOSTOLANSKÝ

V tomto článku sa robí pokus presne definovať syntax a sémantiku jazyka SNOBOL I, vytvoreného na základe myšlienok uvedených v popise jazyka SNOBOL. Jazyk patrí do skupiny tzv. „symbol-manipulation" jazykov.

Syntax jazyka je definovaný pomocou metajazykových premenných. Za každou syntaktickou definíciou nasleduje popis sémantiky a príklady.

## 0. ÚVOD

Oblasť, v ktorej sa vykonáva mechanizácia rôznych druhov ľudskej činnosti pomocou samočinných počítačov, sa stále rozširuje. Samočinné počítače sa začínajú používať k riešeniu problémov, ktoré patria do oblasti mechanického prekladania, matematickej linquistiky, simulácie ľudského poznávacieho procesu a pod. Algoritmické procesy, ktoré sa vyskytujú pri riešení týchto problémov, sú kvalitatívne odlišné napr. od algoritmov vedecko-technických úloh. Možno povedať, že riešenie týchto problémov je závislé na samočinných počítačoch, a preto až s objavením samočinných počítačov sa seriózne pristupuje k formulovaniu úloh tohto typu. Ak sa vyjadrujeme termínami teórie algoritmov, tak pri problematike tohto typu sa v podstate jedná o realizáciu konštrukčných abecedných operátorov v plnej šírke, čo sa týka aj formy údajov, ktoré sa majú spracovať. Zdá sa, že voľba charakteru informácie nerobí ťažkosti, pretože je prirodzené voliť údaje, ktoré pri týchto problémoch sa spracovávajú vo forme reťazcov symbolov, tj. textov. Problém je však v určení základných operácií nad textami. Na základe uvedených skutočností nie je preto prekvapením, že doteraz neboli ustálené základné operácie nad textami, nebola zvolená jednotná symbolika pri popise algoritmov týchto úloh a zrejme potrvá ešte dlhší čas, kým sa toto ustáli, ak to vôbec v plnej šírke problematiky bude možné. Preto algoritmické jazyky, určené pre popis algoritmov úloh tohto typu, sa navzájom líšia a v podstate sú pokusom pre definovanie základných operácií nad textami a voľbu vhodnej symboliky. Z nich najznámejšie sú SLIP, LISP, COMIT a IPL [3].

Pre podobné účely bol vytvorený aj jazyk SNOBOL. Popis tohto jazyka, ktorý bol uverejnený v Journal of the ACM [2], nie je však dostatočne úplný a jasný.

V tejto práci sa robí pokus presne definovať syntax a definovať sémantiku jazyka, ktorý nazveme SNOBOL I, vytvoreného na základe hlavných myšlienok, uvedených v spomenutom popise jazyka SNOBOL.

Jazyk SNOBOL I je určený na popis algoritmických procesov, v ktorých ako údaje vystupujú reťazce symbolov. Teda pomocou tohto jazyka je možné popísať napr. algoritmy úloh z nasledovných oblastí:

1. syntaktická analýza výrazov formálnych jazykov s udanou gramatikou;
2. prekladanie z jedného formálneho jazyka do druhého;
3. hľadanie, rozpoznávanie a zmena údajov alfanumerických;
4. riešenie rôznych problémov z oblasti matematickej linguistiky a pod.

Jazyk SNOBOL I nie je určený pre popis algoritmov numerických problémov, pre ktoré sa používa napr. ALGOL 60, podobne ani pre popis algoritmov, spojených s hromadným spracovaním dát, kde sa používa napr. COBOL.

Ukazuje sa, že z problematiky, pre ktorú je určený jazyk SNOBOL I, vyplýva, že úkony, ktoré budú potrebné k algoritmizácii týchto problémov, možno reprezentovať nasledujúcimi operáciami nad reťazcami:

a) formovanie reťazcov;

b) analýza reťazcov s cieľom určenia istých vlastností analyzovaných reťazcov;

c) zmena reťazcov na základe predchádzajúcej analýzy.

Realizácia týchto operácií tvorí jadro jazyka SNOBOL I. Syntax jazyka, tj. tvorenie textov jazyka, je definovaný pomocou metajazykových premenných ako napr. v ALGOLe [1].

Za každou syntaktickou definíciou nasleduje popis sémantiky, tj. aký význam je priradený správnym textom jazyka pri popise algoritmických procesov a príklady.

Na konci práce sa popisuje algoritmus, ktorý vykonáva syntaktickú kontrolu a „opravu" jednoduchých algolovských aritmetických výrazov, ako napr. $(a + (b + 10 . 7) \times r) \times k$, kde $a$, $b$, $r$, $k$ sú identifikátory.

## 1. ŠTRUKTÚRA JAZYKA

Základný pojem, používaný pre popis v úvode spomenutých algoritmických procesov (v dalšom pod algoritmickým procesom budeme rozumieť proces tohto typu), je reťazcový výraz. Jeho zložkami sú reťazce, reťazcové premenné a obmedzovače.

K tomu, aby bolo možné vyjadriť algoritmický proces, sú pridané príkazy skoku a logické premenné, pomocou ktorých možno vykonať opakovanie niektorej časti procesu, alebo uskutočniť jeho vetvenie.

Jednotlivé etapy algoritmického procesu sú vyjadrené pomocou príkazov programu, ktorý vo všeobecnosti môže byť formovaný z reťazcového výrazu, príkazu priradenia a príkazu skoku. Realizácia algoritmického procesu vyžaduje, aby jeden príkaz programu sa odvolával na druhý, preto príkazy programu môžu byť opatrené náveštiami.

Program je postupnosť príkazov programu. Účinok programu, tj. proces, ktorý prebieha pri jeho vykonávaní, môže byť odvodený z programu pomocou syntaktickej analýzy programu a aplikácie odpovedajúcej sémantiky syntaktických jednotiek. V dalšom bude definovaný syntax a sémantika jazyka.

⟨základný symbol⟩ ::= ⟨prvok abecedy⟩|⟨logická hodnota⟩|⟨obmedzovač⟩

### 2.1. Prvok abecedy*

⟨prvok abecedy⟩ ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n |
o | p | q | r | s | t | u | v | w | x | y | z | A |
B | C | D | E | F | G | H | I | J | K | L | M | N | O |
P | Q | R | S | T | U | V | W | X | Y | Z | 0 | 1 | 2 |
3 | 4 | 5 | 6 | 7 | 8 | 9 | , | . | ; | ( | ) |

: | := | 10 | + | − | × | / | ÷ | ↑ | < | ≦ | = | ≧ |
> | ≠ | ≡ | ⊃ | ∨ | ∧ | ¬ |

**go to** | **if** | **then** | **else** | **for** | **do** | **step** | **until** | **while** |
**comment** | **begin** | **end** | **own** | **Boolean** | **integer** | **real** |
**array** | **switch** | **procedure** | **string** | **label** | **value** |
**false** | **true**

Z prvkov abecedy sa formujú reťazce. Množina prvkov abecedy môže byť vhodne rozšírená, resp. zúžená novými rozpoznateľnými prvkami.

### 2.2. Logické hodnoty

⟨logická hodnota⟩::= **TRUE** | **FALSE**

Logické hodnoty majú zrejmý, vžitý význam.

### 2.3. Obmedzovače

⟨obmedzovač⟩ ::= * | ' | ≐ | ⊔ | ⌋ | & | ⌿ | : | ⌈ | **T** | **F**

Obmedzovače sa používajú pri konštrukcii reťazcových výrazov, príkazov priradenia a príkazov skoku. Ich bližší význam bude objasnený na patričnom mieste.

* Polotučná tlač sa používa k určeniu nezávislých prvkov abecedy.

# 3. REŤAZCE A HODNOTY

## 3.1. Reťazce

### 3.1.1. *Syntax*

⟨prázdny reťazec⟩ ::=

⟨plný reťazec⟩ ::= ⟨prvok abecedy⟩|⟨plný reťazec⟩ ⟨prvok abecedy⟩

⟨reťazec⟩ ::= ⟨prázdny reťazec⟩|⟨plný reťazec⟩

### 3.1.2. *Príklady reťazcov*

11

aaa

01.25

26a0bb

Dr. Bogner

**begin real** a, b; a := = 2.3; b := a ↑ 3 **end**

((a × (b + c) − 2 × z) − k) × a ↑ 2

### 3.1.3. *Sémantika*

Hodnota reťazca je reťazec sám. Dĺžka reťazca je počet prvkov abecedy v reťazci. Ak $\alpha_1$ je reťazec, potom jeho dĺžku budeme označovat $\lambda\alpha_1$.

## 3.2. Hodnoty

Hodnota je číslo, reťazec alebo logická hodnota. Niektoré synaktické jednotky nadobúdajú hodnôt, ktoré sa všeobecne menia počas vykonávania programu. Hodnoty reťazcového výrazu a jeho zložiek sú definované v kap. 4.1.3 a 4.2.3.

# 4. VÝRAZY

Reťazcové výrazy sú v jazyku prvotnými zložkami programov popisujúcich algoritmické procesy. Zložkami týchto výrazov sú reťazce, reťazcové premenné a obmedzovače.

4.1.1. *Syntax*

⟨znak identifikátora⟩ ::= α | β | γ | δ | ε | ζ | η | ϑ | ι | ϰ | λ |
μ | ν | ξ | π | ϱ | σ | τ | υ | φ | χ | ψ | ω |

⟨identifikátor⟩ ::= ⟨znak identifikátora⟩ ⟨reťazec⟩

⟨číslica⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

⟨číslo⟩ ::= ⟨číslica⟩ | ⟨číslo⟩ ⟨číslica⟩

⟨označenie dĺžky⟩ ::= ⟨číslo⟩ | ⟨identifikátor⟩

⟨reťazcová premenná pevnej dĺžky⟩ ::= ⟨identifikátor⟩ ↖ ⟨označenie dĺžky⟩

⟨reťazcová premenná⟩ ::= ⟨identifikátor⟩ | ⟨reťazcová premenná pevnej dĺžky⟩

4.1.2. *Príklady*

Identifikátory:

α zoznam
β A 1
γ 1
ε chyba

Reťazcová premenná pevnej dĺžky:

δ sum ↖ 7

φ funkcia ↖ β argument

4.1.3. *Sémantika*

Reťazcová premenná označuje hodnotu. Táto hodnota môže byť použitá vo výrazoch k vytvoreniu iných hodnôt. Môže byť zmenená príkazom priradenia.

Reťazcová premenná pevnej dĺžky označuje reťazec s udanou dĺžkou. Táto je daná textom za symbolom ↖. Ak tento text je číslo, tak je to desiatkový zápis dĺžky reťazcovej premennej. Ak je to identifikátor, tak jeho hodnota určuje dĺžku reťazcovej premennej. Reťazcová premenná pevnej dĺžky vystupuje iba pri konštrukcii vzoru (pozri kap. 5.1.1). V reťazcových výrazoch sa používa len jej identifikátor.

4.2.1. *Syntax*

**Reťazcový výraz**

⟨operand⟩ ::= ⟨reťazec⟩ | ⟨identifikátor⟩

⟨reťazcový výraz⟩ ::= ⟨operand⟩ | ⟨reťazcový výraz⟩ & ⟨operand⟩

4.2.2. *Príklady*

Reťazcový výraz:

Bogner, 22. 12. 1922

α kvalifikácia

β prax

Bogner, 22. 12. 1922 & α kvalifikácia & β prax

4.2.3. *Sémantika*

Reťazcový výraz je pravidlo pre získanie reťazcovej hodnoty. Toto získanie sa uskutoční vykonaním operácie zreťazenia na skutočných reťazcových hodnotách. Operácia zreťazenia formuje novú reťazcovú hodnotu, ktorá vznikne zreťazením reťazcových hodnôt operandov. (Ak $\alpha_1$, $\alpha_2$ sú reťazce, výsledok operácie $\alpha_1$ & $\alpha_2$ je reťazec $\alpha_1\alpha_2$. Operácia zreťazenia je asociatívna.) Reťazcová hodnota je zrejmá v prípade operandu reťazca. V prípade reťazcovej premennej je to posledná priradená hodnota tejto premennej.

## 5. PRÍKAZY

Jednotky jazyka, ktoré majú operačný význam, sa nazývajú *príkazy*. V uvažovanom jazyku vystupujú príkazy priradenia a skoku, ktoré sú zložkami príkazov programu. Normálne sa príkazy programu vykonávajú postupne jeden za druhým. Riadiaca časť príkazu programu môže prerušiť tento postupný spôsob vykonávania príkazov programu tak, že určí, ktorý príkaz sa bude vykonávať ako nasledujúci. Aby bolo možné určiť poradie vykonávania príkazov programu, môžu byť tieto opatrené návešťami.

### 5.1. Príkazy priradenia

5.1.1. *Syntax*

⟨jednoduchý vzor⟩ ::= ⟨reťazcový výraz⟩ * ⟨reťazcová premenná⟩ *
⟨reťazcový výraz⟩

⟨vzor prvého druhu⟩ ::= ⟨jednoduchý vzor⟩|⟨vzor prvého druhu⟩ ⁄ ⟨jednoduchý vzor⟩

⟨vzor⟩ ::= ⟨reťazcový výraz⟩ | ⟨vzor prvého druhu⟩

⟨priradnie podľa vzoru⟩ ::= ⟨identifikátor⟩ ⌋ ⟨vzor⟩

⟨zámena⟩ ::= ⟨priradenie podľa vzoru⟩ ≙ ⟨reťazcový výraz⟩

⟨prvok ľavej časti⟩ ::= ⟨identifikátor⟩ :≙

⟨ľavá časť⟩ ::= ⟨prvok ľavej časti⟩ | ⟨ľavá časť⟩ ⟨prvok ľavej časti⟩

⟨jednoduché priradenie⟩ ::= ⟨ľavá časť⟩ ⟨reťazcový výraz⟩

⟨príkaz priradenia⟩ ::= ⟨jednoduché priradenie⟩ | ⟨zámena⟩ | ⟨priradenie podľa vzoru⟩

5.1.2. *Príklady*

Jednoduchý vzor:

Bogner, ∗ β data ∗ α kvalifikácia
**if** ∗ α výraz ⧹ 3 ∗ **then**

Vzor:

a ↑ b
(∗ ε zbytok ∗) ⁄ ∗ γ op ⧹ 1 ∗ (
**begin** ∗ α úsek ∗;

Priradenie podľa vzoru:

α blok ⌋ **begin** ∗ α usek ∗;
φ vyraz ⌋ (∗ ε zbytok ∗) ⁄ ∗ γ op ⧹ 1 ∗ (

Zámena:

φ vyraz ⌋ (∗ ε zbytok ∗) ≙ A
γ vyraz ⌋ a + b ≙ B
α AZ ⌋ E ≙ α B & α K & 22

Jednoduché priradenie:

α Al :≙ α A2 :≙ α A3 :≙ 111 a 222
β Bl :≙ β B2 :≙ β meno & β data & 357

Príkazov priradenia sa používa k priradeniu hodnoty jednej alebo viacerým premenným. U jednotlivých druhov príkazov priradenia prebieha proces priradenia nasledovne:

a) *Jednoduché priradenie.* Určí sa hodnota reťazcového výrazu v príkaze a priradí sa všetkým premenným v ľavej časti.

b) *Priradenie podľa vzoru.* Vzor v príkaze, v súhlase so syntaxom, môže byť tvorený postupnosťou jednoduchých vzorov alebo reťazcovým výrazom. V prvom prípade vzor má teda tvar

$$\alpha_1 * \beta_1 * \gamma_1 \swarrow \alpha_2 * \beta_2 * \gamma_2 \swarrow \ldots \swarrow \alpha_n * \beta_n * \gamma_n , \tag{1}$$

kde $\alpha_1, \alpha_2, \ldots, \alpha_n, \gamma_1, \gamma_2, \ldots, \gamma_n$ sú reťazcové výrazy a $\beta_1, \ldots, \beta_n$ sú reťazcové premenné.

Nech $A$ je reťazec, ktorý je hodnotou identifikátora v príkaze. Nech $A_1, A_2, \ldots, A_n$ a $C_1, C_2, \ldots, C_n$ sú postupne hodnoty reťazcových výrazov $\alpha_1, \alpha_2, \ldots, \alpha_n$ a $\gamma_1, \ldots, \gamma_n$. Priradenie podľa vzoru prebieha nasledovne: Ak sú splnené vo všeobecnosti dve podmienky:

1. Existujú také reťazce $B_i$ $(1 \leqq i \leqq n)$ a $D_j$ $(1 \leqq j \leqq n + 1)$, že reťazec $A$ je hodnotou reťazcového výrazu

$$D_1 \,\&\, A_1 \,\&\, B_1 \,\&\, C_1 \,\&\, D_2 \,\&\, A_2 \,\&\, B_2 \,\&\, C_2 \,\&\, \ldots \,\&\, D_n \,\&\, A_n \,\&\, B_n \,\&\, C_n \,\&\, D_{n+1} , \tag{2}$$

pričom $2n$-ica reťazcov $D_1, B_1, D_2, B_2, \ldots, D_n, B_n$ má najmenšiu dĺžku* zo všetkých $2n$-íc, ktoré môžu vystupovať pri konštrukcii výrazu (2) tak, že reťazec $A$ je hodnotou (2).

2. Ak vo vzore

$$\beta_{i_1}, \beta_{i_2}, \ldots, \beta_{i_z}$$

sú reťazcové premenné pevnej dĺžky o dĺžke

$$L_1, L_2, \ldots, L_z \quad \text{a} \quad \lambda B_{i_1} = L_1 , \quad \lambda B_{i_2} = L_2 , \quad \lambda B_{i_z} = L_z ,$$

potom premenným vo vzore $\beta_1, \ldots, \beta_n$ sa priradia postupne hodnoty

$$B_1, B_2, \ldots, B_n .$$

V opačnom prípade sa priradenie nevykoná.

* Nech $A$ je množina $n$-íc reťazcov. Hovoríme, že $n$-ica reťazcov $(\alpha_1, \alpha_2, \ldots, \alpha_n) \in A$ má menšiu dĺžku, ako $n$-cia $(\beta_1, \beta_2, \ldots, \beta_n) \in A$, ak pre prvé $i$ $(1 \leqq i \leqq n)$, pre ktoré $\lambda\alpha_i \neq \lambda\beta_i$, je $\lambda\alpha_i < \lambda\beta_i$.

Ak vzor v príkaze má tvar reťazcového výrazu, potom účinok príkazu sa redukuje na určenie hodnoty reťazcového výrazu a zistenia, či sa vyskytuje ako podreťazec v reťazci, ktorý je hodnotou identifikátora v príkaze. Výsledok sa môže použiť v príkaze skoku (pozri kap. 5.2.3).

c) *Zámena.* Je rozšírením príkazu priradenia podľa vzoru. Ak príkaz priradenia podľa vzoru, ktorý je jej časťou, bol úspešný, tj. ak ku všetkým premenným vo vzore sa priradila hodnota, resp. určil sa zľava prvý výskyt hodnoty reťazcového výrazu, ktorý bol vzorom, potom vykoná nasledujúcu zámenu:

Vo vzore sa symboly * nahradia symbolmi &, čím sa získajú reťazcové výrazy (vzájomne sú oddelené obmedzovačom |). Postupne prvý výskyt hodnôt týchto reťazcových výrazov v reťazci, ktorý je hodnotou identifikátora v priradení podľa vzoru, sa nahradí hodnotou reťazcového výrazu vpravo od obmedzovača ≙.

Príkazom priradenia podľa vzoru a zámenou sa získajú logické hodnoty **TRUE** a **FALSE**, ktoré sa používajú v príkaze skoku (5.2.3).

5.1.4. *Prázdny príkaz*

⟨prázdny príkaz⟩ ::=

Prázdny príkaz nemá žiadny operačný účinok. Používa sa pri konštrukcii príkazu skoku.

## 5.2. Príkazy skoku

5.2.1. *Syntax*

⟨priame náveštie⟩ ::= ⟨reťazec⟩

⟨nepriame náveštie⟩ ::= ⟨identifikátor⟩

⟨náveštie⟩ ::= ⟨priame náveštie⟩|⟨nepriame náveštie⟩

⟨označenie logickej hodnoty⟩ ::= Γ ⟨identifikátor⟩

⟨nepodmienený príkaz skoku⟩ ::= Ł ⟨náveštie⟩

⟨podmienený príkaz skoku⟩ ::= **T** ⟨náveštie⟩|**F** ⟨náveštie⟩

⟨zložený príkaz skoku⟩ ::= ⟨podmienený príkaz skoku⟩ ⟨nepodmienený príkaz skoku⟩

⟨príkaz skoku prvého druhu⟩ ::= ⟨podmienený príkaz skoku⟩|⟨zložený príkaz skoku⟩

⟨príkaz skoku⟩ ::= ⟨nepodmienený príkaz skoku⟩|⟨označenie logickej hodnoty⟩
⟨príkaz skoku prvého druhu⟩|⟨označenie logickej hodnoty⟩
⟨nepodmienený príkaz skoku⟩|⟨prázdny príkaz⟩|⟨označenie logickej hodnoty⟩

5.2.2. *Príklady*

Nepodmienený príkaz skoku:

Ł SUM

Ł α ITER

Príkaz skoku:

Ł SUM

**T** α NA Ł SUM

Γ β OLHT **F** α SOS Ł SUM

Podmienený príkaz skoku:

**T** SUM

**F** ITER

Zložený príkaz skoku:

**T** α SOS Ł SUM

**F** ITER Ł VOLBA

5.2.3. *Sémantika*

Nepodmienený príkaz skoku spôsobí, že ako nasledujúci príkaz programu sa bude vykonávať ten, ktorý má to isté náveštie, aké je v príkaze skoku.

Účinok podmieneného príkazu skoku závisí od hodnoty označenia logickej hodnoty. Predpokladá sa, že označeniu logickej hodnoty sa priradila niektorá z hodnôt **TRUE** alebo **FALSE** pred vykonaním podmieneného príkazu skoku. Ak hodnota je **TRUE**, potom účinok podmieneného príkazu skoku **T** ⟨náveštie⟩ je rovný účinku nepodmieneného príkazu skoku Ł ⟨náveštie⟩ a účinok príkazu **F** ⟨náveštie⟩ sa rovná účinku prázdneho príkazu. Pri hodnote **FALSE** majú tieto príkazy opačný efekt. Pri zloženom príkaze skoku sa najprv vyhodnocuje zložka podmieneného príkazu popísaným spôsobom. Ak má účinok nepodmieneného príkazu skoku, tento sa rovná účinku celého zloženého príkazu skoku. V opačnom prípade účinok zloženého príkazu skoku sa rovná vykonaniu nepodmieneného príkazu skoku, ktorý v ňom vystupuje.

Hodnota označenia logickej hodnoty závisí na účinku príkazu priradenia. Ak sa vykonalo priradenie podľa vzoru, alebo zámena, resp. sa určil prvý výskyt reťazca, ktorý je hodnotou reťazcového výrazu reprezentujúceho vzor, získa sa logická hodnota **TRUE**, ktorá sa priradí označeniu logickej hodnoty, ktoré vystupuje v tomto príkaze programu.

Ak príkaz skoku obsahuje iba označenie logickej hodnoty, potom jeho účinok sa redukuje na priradenie hodnoty **TRUE** alebo **FALSE** tomuto označeniu logickej hodnoty spôsobom, popísaným v predchádzajúcom odstavci.

V súhlase so syntaxom príkaz skoku môže mať tvar podmieneného alebo zloženého príkazu skoku bez označenia logickej hodnoty. Potom jeho účinok závisí od logickej hodnoty, získanej príkazom priradenia v tomto príkaze programu.

Význam priameho náveštia je zrejmý. Pri nepriamom náveští posledná hodnota (v dynamickom zmysle) udanej reťazcovej premennej je náveštie patriace k príkazu skoku.

Reťazcová premenná volená ako označenie logickej hodnoty môže nadobúdať iba dve logické hodnoty, **TRUE** alebo **FALSE**. 

### 5.3. Príkazy programu

5.3.1. *Syntax*

⟨príkaz programu bez náveštia⟩ ::= ⟨príkaz priradenia⟩|⟨príkaz skoku⟩|⟨príkaz priradenia⟩ ⟨príkaz skoku⟩

⟨príkaz programu⟩ ::= ⟨príkaz programu bez náveštia⟩|⟨priame náveštie⟩ ⟨príkaz programu bez náveštia⟩

5.3.2. *Príklady*

Príkazy programu bez náveštia:

```
α x :≙ 0 1 2 3
β N :≙
α K ⌋ * α D \ 1 *    F END
L Z ⌋ 3 2 0 1        ⌐ α NA ⊧ SUM
```

Príkazy programu:

```
PR 1 α x := 0 1 2 3
PR 2 β P := β N & α D    ⊧ PR 3
    β M ⌋ α D * β M *    ⊥ β L 1
```

5.3.3. *Sémantika*

Účinok príkazu programu je daný vykonaním príkazu priradenia a príkazu skoku, ktoré tvoria príkaz programu.

Časove sa prvý vykonáva príkaz priradenia. Ak sa jeho vykonaním získa logická hodnota, táto sa v nasledujúcom môže priradiť označeniu logickej hodnoty. Potom sa vykoná vlastný príkaz skoku.

### 5.4. Program

5.4.1. *Syntax*

Program tvorí postupnosť príkazov programu. Presná definícia syntaxu:

⟨program⟩ ::= ⟨príkaz programu⟩|⟨program⟩:⟨príkaz programu⟩

Účinok programu spočíva v postupnom vykonávaní príkazov programu.

## 6. PRÍKLADY

**6.1.** Nech $\alpha_1, \alpha_2, \ldots, \alpha_n$ sú symboly s klesajúcou prioritou (tedy $\alpha_1$ má najväčšiu a $\alpha_n$ 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 $\alpha_{j_i}$ (pre $1 \leqq i < k$) je väčšia ako priorita $\alpha_{j_{j+1}}$. Program pre realizáciu tohto procesu je nasledovný:

```
        β P :≐ α_i1, α_i2, ..., α_ik:
        β N :≐ :
        β M := α_1, α_2, ..., α_n:
PR 1    β M ⌋ * α D \ 1 * , ≐   F END:
PR 2    β P ⌋ α D ≐             F PR 1:
PR 3    β N ⌋ :≐ β N & α D      Ł PR 2:
END
```

**6.2.** Treba vykonať syntaktickú kontrolu častí jednoduchých algolovských aritmetických výrazov, syntax ktorých je definovaná nasledovne:

⟨operand⟩ ::= ⟨jednoduchá premenná⟩|⟨číslo bez znamienka⟩
(⟨jednoduchý aritmetický výraz⟩)

⟨jednoduchý aritmetický výraz⟩ ::= ⟨operand⟩|⟨operand⟩
⟨operátor⟩ ⟨operand⟩|
⟨jednoduchý aritmetický výraz⟩
⟨operátor⟩ ⟨operand⟩

Definície jednoduchej premennej, čísla bez znamienka a operátoru sú zhodné s definíciami týchto metajazykových premenných v ALGOLe [1].

Algoritmus syntaktickej kontroly sa skladá z dvoch častí, ktoré vyžadujú dva prechody cez jednoduchý aritmetický výraz.

Pri prvom prechode, ak jednoduchý aritmetický výraz obsahuje zátvorky, sa vykonáva kontrola vzhľadom na zátvorky. Táto spočíva v postupnom hľadaní odpovedajúcej ľavej zátvorky k pravej alebo naopak. Ak niektorá ľavá (pravá) zátvorka nemá odpovedajúcu pravú (ľavú) zátvorku, tak nakoniec (na začiatok) jednoduchého aritmetického výrazu sa „pripíše" pravá (ľavá) zátvorka.

Ak jednoduchý aritmetický výraz neobsahuje zátvorky, tak pri druhom prechode sa zisťuje, či tento výraz je tvorený postupnosťou operand, operátor, ..., operand.

Ak ho netvorí takáto postupnosť, tak sa vykonávajú opravy, ktoré nakoniec výrazu dajú tvar požadovanej postupnosti.

Keď výraz obsahuje zátvorky, tak pri druhom prechode sa v jeho „upravenom" tvare (tento môže byť zhodný s pôvodným), ktorý sa získal pri prvom prechode, zisťuje, či každá uzátvorkovaná časť je tvorená postupnosťou operand, operátor, ..., operand. Ak nie, tak sa vykonávajú opravy, ktoré tejto časti dajú tvar požadovanej postupnosti. Opravy vykonané v oboch prechodoch formujú nový reťazec.

Pri popise algoritmu predpokladáme, že kontrolovaný jednoduchý aritmetický výraz je hodnotou identifikátora α A. Ďalej, že hodnota identifikátora α AB je reťazec vytvorený z abecedy v ALGOLe, α CY má hodnotu, ktorá je reťazcom vytvoreným z desiatkových číslic, hodnota reťazca α OP je reťazec vytvorený z aritmetických operátorov a reťazec . $_{10}$ je hodnotou identifikátora α ROZ.

## 7. POPIS ALGORITMU

```
P  1    α B :≐ α A ⋮
P  2    α ZTV := ⋮
P  3    α B ⌋ * α ZBYTOK * )                    F P 4 Ł P 7 ⋮
P  4    α B ⌋ (≐ ⋮
P  5    α ZVT :≐ α ZVT & ) ⋮
P  6    α A :≐ α A &                            Ł P 4 ⋮
P  7    α B ⌋ (                                 T P 11 ⋮
P  8    α B ⌋ ) ≐                               F P 17 ⋮
P  9    α ZVT :≐ ( & α ZVT ⋮
P 10    α A :≐ ( & α A                          Ł P 8 ⋮
P 11    α ZBYTOK⌋)                              F P 14 ⋮
P 12    α ZBYTOK ⌋ ( * α ZBYTOK *               T P 12 ⋮
P 13    α B ⌋ ( & α ZBYTOK & ) ≐ A              Ł P 3 ⋮
P 14    α ZTV :≐ ( & α ZTV ⋮
P 15    α A :≐ ( & α A ⋮
P 16    α B ⌋ α ZBYTOK & ) ≐ A                  Ł P 3 ⋮
P 18    α A ⌋ * α ZBYTOK * )                    F P 67 Ł P 70 ⋮
P 19    α ZBYTOK ⌋ ( * α ZBYTOK *               T P 19 ⋮
P 20    β HH :≐ β HODNOTA :≐ α ZBYTOK ⋮
P 21    α ZBYTOK ⌋ * α ZNAK \ 1 * ≐             F P 56 ⋮
P 22    α AB ⌋ α ZNAK                           T P 26 ⋮
P 23    α CY ⌋ α ZNAK                           T P 31 ⋮
P 24    α ROZ ⌋ α ZNAK                          T P 33 ⋮
P 25    β HODNOTA ⌋ α ZNAK ≐ 1                  Ł P 31 ⋮
P 26    α ZBYTOK ⌋ * α ZNAK / 1 * ≐             F α R ⋮
P 27    α AB ⌋ α ZNAK                           T P 26 ⋮
P 28    α CY ⌋ α ZNAK                           T P 26 ⋮
```

```
P 29   α OP ⌋ α ZNAK                                  T P 21 ⋮
P 30   β HODNOTA ⌋ α ZNAK ≟ 1                         Ł P 26 ⋮
P 31   α ZBYTOK ⌋ ∗ α ZNAK ↖ 1 ∗ ≟                    F α R ⋮
P 32   α CY ⌋ α ZNAK                                  T P 31 ⋮
P 33   α ZNAK ⌋ .                                     T P 37 ⋮
P 34   α ZNAK ⌋ 10                                    T P 44 ⋮
P 35   α OP ⌋ α ZNAK                                  T P 21 ⋮
P 36   β HODNOTA ⌋ α ZNAK ≟ 1                         Ł P 31 ⋮
P 37   α ZBYTOK ⌋ ∗ α ZNAK ↖ 1 ∗ ≟                    F P 57 ⋮
P 38   α CY ⌋ α ZNAK                                  F P 72 ⋮
P 39   α ZBYTOK ⌋ ∗ α ZNAK ↖ 1 ∗                      F α R ⋮
P 40   α CY ⌋ α ZNAK                                  T P 39 ⋮
P 41   α ZNAK ⌋ 10                                    T P 44 ⋮
P 42   α OP ⌋ α ZNAK                                  T P 21 ⋮
P 43   β HODNOTA ⌋ α ZNAK ≟ 1                         Ł P 39 ⋮
P 44   α ZBYTOK ⌋ ∗ α ZNAK ↖ 1 ∗ ≟                    F P 58 ⋮
P 45   α ZNAK ⌋ +                                     T P 53 ⋮
P 46   α ZNAK ⌋ −                                     T P 53 ⋮
P 47   α CY ⌋ α ZNAK                                  T P 49 ⋮
P 48   β HODNOTA ⌋ α ZNAK ≟ 1 ⋮
P 49   α ZBYTOK ⌋ ∗ α ZNAK ↖ 1 ∗                      F α R ⋮
P 50   α CY ⌋ α ZNAK                                  T P 49 ⋮
P 51   α OP ⌋ α ZNAK                                  T P 21 ⋮
P 52   β HODNOTA ⌋ α ZNAK ≟ 1                         Ł P 49 ⋮
P 53   α ZBYTOK ⌋ ∗ α ZNAK ↖ 1 ∗ ≟                    F P 59 ⋮
P 54   α CY ⌋ α ZNAK                                  T P 49 ⋮
P 55   β HODNOTA ⌋ α ZNAK ≟ 1                         Ł P 49 ⋮
P 56   α ZBYTOK :≟ 1                                  Ł P 63 ⋮
P 57   α ZBYTOK :≟ 1                                  Ł P 64 ⋮
P 58   α ZBYTOK :≟ 1                                  Ł P 65 ⋮
P 59   α ZBYTOK :≟ 1                                  Ł P 66 ⋮
P 60   α A ⌋ ( & β NH &) ≟ ZÁTVORKA ⋮
P 61   β HH ⌋ β HODNOTA                               T β R ⋮
P 62   α ZTV ≟ α ZTV & HODNOTA                        Ł R ⋮
P 63   β HODNOTA :≟ β HODNOTA & 1                     Ł P 21 ⋮
P 64   β HODNOTA :≟ β HODNOTA & 1                     Ł P 37 ⋮
P 65   β HODNOTA :≟ β HODNOTA & 1                     Ł P 44 ⋮
P 66   β HODNOTA :≟ β HODNOTA & 1                     Ł P 53 ⋮
P 67   α R :≟ P 61 ⋮
P 68   β R :≟ END ⋮
P 69   α ZBYTOK :≟ α A                               Ł P 20 ⋮
P 70   α R :≟ P 60 ⋮
```

```
P 71    β R :≐ P 18                                  Ł P 73 ⋮
P 72    β HODNOTA T α ZNAK ≐ 1                       Ł P 39 ⋮
P 73    α A ⌋ ( & α ZBYTOK                           F P 21 ⋮
P 74    α A ⌋ * α P ↖ 1 * ( & ZBYTOK & )             F P 78 ⋮
P 75    α OP ⌋ α P                                   T P 81 ⋮
P 76    α P ⌋ (                                      T P 81 ⋮
P 77    β HODNOTA :≐ β HODNOTA & +                   Ł P 81 ⋮
P 78    α A ⌋ (& α ZBYTOK & ) * α P1 ↖ 1 *           F P 21 ⋮
P 79    α OP ⌋ α P1                                  T P 21 ⋮
P 80    β HODNOTA :≐ β HODNOTA & *                   Ł P 21 ⋮
P 81    α A ⌋ (& α ZBYTOK & ) * α P1 ↖ 1 *           F P 21 ⋮
P 82    α OP ⌋ α P1                                  T P 21 ⋮
P 83    α P1 ⌋ )                                     T P 21 ⋮
P 84    β HODNOTA :≐ HODNOTA & × ⋮
END
```

(Došlo dňa 28. júla 1966.)

LITERATÚRA

[1] J. W. Backus: Report on the Algorithmic Language ALGOL 60. Numerische Mathematik 2 (1960).

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

[3] 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).

SUMMARY

## The Definition of the Syntax and Semantics of Language SNOBOL I

EDUARD KOSTOLANSKÝ

In the article an attempt is made to define precisely the syntax and semantics of the language SNOBOL I constructed on the basis of ideas listed in the description of the language SNOBOL. The language belongs to the group of the so called "symbol-manipulation" languages.

The basic notion used to describe algorithmic processes, which is recordable in SNOBOL I, is the chain expression. Its components are chains, chain variables and restrictors.

To make the expression of an algorithmic process possible, step instructions and logical variables are added, by means of which the repetition of some part of the process or its branching may be carried out.

The single sections of the algorithmic process are expressed by means of program instructions which generally may be formed of the chain expression, the instructions of sequence and jump. The performance of the algorithmic process requires that one program instruction refers to the other, these therefore may be equipped with signalling devices.

The program is a sequence of program instructions. The effect of the program, i.e. the process running during its performance, may be derived from the program by means of a syntactical analysis of the program and by application of the relevant semantics of syntactical units. The syntax of the language is defined by means of meta-linguistic variables. Each symbolic definition is followed by a description of semantics and by examples.

*Eduard Kostolanský, prom. mat., Ústav technickej kybernetiky SAV, Bratislava, Dúbravská cesta.*