Jiří Kopřiva

Syntaktická analýza textu

Pokroky matematiky, fyziky a astronomie, Vol. 11 (1966), No. 5, 265--287

Persistent URL: http://dml.cz/dmlcz/137753

POKROKY MATEMATIKY, FYZIKY A ASTRONOMIE
ROČNÍK XI — ČÍSLO 5

SYNTAKTICKÁ ANALÝZA TEXTU

Jiří Kopřiva, Brno

ÚVOD

Při *automatizaci programování* pro samočinné počítače se snažíme přenést na počítací stroj co největší část mechanické, netvůrčí práce, nutné před provedením samotného výpočtu. Jinak řečeno, snažíme se ponechat člověku pouze volbu či vytvoření vhodného postupu řešení (např. numerické metody pro řešení diferenciální rovnice) a umožňujeme mu, aby zapsal tento postup způsobem, který je víceméně blízký způsobu zápisu řešení používanému v matematice a formální logice.

Pro zápis programu, který je určen k řešení úloh různých druhů, byla vytvořena celá řada tzv. *programovacích jazyků.* Jsou to umělé formální jazyky, z nichž každý je vhodný pro formulaci algoritmů řešících úlohy jistých typů. O některých otázkách s tím spojených viz např. článek [1] v tomto časopise. Předložíme-li modernímu samočinnému počítači program zapsaný ve vhodném programovacím jazyku, musí si ho před provedením výpočtu „přeložit" do své „řeči". To znamená, že jednotlivé příkazy dodaného programu si vyjádří příslušnou posloupností elementárních operací. Po umístění přeloženého programu ve své paměti a po přijetí potřebných vstupních dat je počítač schopen provést výpočet a vydat (např. vytisknout) výsledek.

Při sestavování programu jak v programovacím jazyku, tak v kódu samočinného počítače se musíme řídit zásadami, které se dají formálně vyjádřit, zformalisovat. Přepis programu z programovacího jazyka do kódu počítače vlastně znamená překládání textů z jednoho formálního jazyka do druhého. Tento postup, který nazýváme zpravidla *kompilací,* provádí počítač na základě jednou provždy sestaveného programu, tzv. *kompilátoru.*

Proces kompilace má zhruba dvě fáze: a) *rozbor původního (vstupního) textu*; b) *sestavení hledaného překladu (výstupního textu).* Nesmíme si ovšem představovat, že tyto dvě hlavní fáze probíhají odděleně v časové následnosti. Zpravidla probíhá kompilace ve více fázích následujících po sobě. V každé z nich se jednak upravuje vstupní text, jednak se připravují prvky výstupního textu. To znamená, že v každé fázi se znova prochází upraveným vstupním textem nebo jeho částí. K tomu přistupuje *kontrola správnosti vstupního textu.* Při ní kompilátor zjišťuje, zda byl tento text sestaven v souhlase s formálními (syntaktickými) i sémantickými (významovými) zásadami programovacího jazyka.

Nemůžeme se zde zabývat otázkou kompilace programovacích jazyků v celé její šíři. Všimneme si blíže hlavně jen některých formálních zásad a postupů, které se uplatňují při rozboru textů psaných formálním programovacím jazykem, tj. při tzv. *syntaktické analýze.* K tomu ovšem potřebujeme být aspoň částečně poučeni o popisu formálních jazyků a o syntéze frází v nich. Těmto otázkám věnujeme následující druhý odstavec. V dalších odstavcích popíšeme některé způsoby syntaktické analýzy, která do jisté míry odpovídá rozboru vět živých jazyků, jak se provádí při vyučování mateřštiny na školách nižších stupňů. Zároveň si ukážeme, že při vhodném spojení syntaktických pravidel vstupního jazyka s překládacími pravidly můžeme v průběhu syntaktické analýzy současně vytvářet výstupní text.

POPIS A SYNTÉZA FORMÁLNÍCH JAZYKŮ

Text, který je vytvořen správně podle zásad syntaxe formálního jazyka, budeme nazývat *frází* tohoto jazyka. Je to posloupnost symbolů tzv. *terminální abecedy* jazyka neboli *terminálních symbolů.* Terminální abeceda je konečná množina symbolů vhodných pro vyjádření procesů (vlastně jejich algoritmů), které chceme popsat frázemi jazyka. Tak např. terminální symboly dnes velmi rozšířeného programovacího jazyka ALGOL 60 (viz [1], str. 161, a [3]) jsou mimo jiné malá a velká písmena latinské abecedy, desítkové cifry, značky pro aritmetické a logické operace, různé omezovače (čárka, tečka, středník, dvojtečka, závorky apod.), ale také vybraná anglická slova (považovaná ovšem ze jeden symbol), např. **begin, else, false, true** atd.

Každou konečnou posloupnost znaků nějaké dané abecedy nazýváme *řetězem* nad touto abecedou. Pouhým kladením řetězů vedle sebe, tzv. *zřetězováním*, získáváme opět řetězy nad touž abecedou. Množina všech řetězů nad konečnou abecedou tvoří tedy tzv. volnou pologrupu, kde grupovou operací je právě zřetězování. Tato pologrupa má jednotku a je jí tzv. *prázdný řetěz.* Je to řetěz neobsahující žádný symbol. Nazveme-li *délkou* řetězu počet jeho symbolů, má prázdný řetěz délku nula. Např. tedy prázdný řetěz, $ab//a$, $bb+$ jsou řetězy nad abecedou $\{a, b, +, /\}$, ale $c/+a$ není řetěz nad touto abecedou.

K tomu, abychom mezi všemi řetězy nad danou abecedou terminálních symbolů formálního jazyka uměli vybrat ty, které jsou jeho frázemi, popř. abychom tyto fráze uměli vytvářet, musíme mít nějaká pravidla. Ukazuje se, že skladbu frází některých programovacích jazyků lze do jisté míry popsat způsobem používaným pro popis *bezkontextových frázových jazyků* (viz [2]). *Gramatiku* takového jazyka tvoří konečný počet *přepisovacích pravidel* tvaru

$$A \to \alpha$$

(čteme A lze přepsat jako α). Na levé straně znaku $\to$ stojí tzv. *metalingvistická proměnná*, na pravé straně je neprázdný řetěz nad abecedou, která je sjednocením terminální abecedy a (konečné) množiny metalingvistických proměnných. Metaling-

vistická proměnná (krátce také metaproměnná) odpovídá zhruba gramatické kategorii živého jazyka. Můžeme si pod ní představovat množinu jistých řetězů nad terminální abecedou vytvořených způsobem, který hned popíšeme. Každý řetěz z této množiny považujeme za *hodnotu* uvažované metaproměnné. Budeme v tomto odstavci označovat terminální symboly malými latinskými písmeny, metaproměnné velkými latinskými písmeny a řetězy (složené obecně ze symbolů obojího druhu) malými řeckými písmeny.

Řekneme, že řetěz ψ (složený obecně z terminálních symbolů a metaproměnných) je *bezprostředně generován* jiným takovým řetězem φ (znak $\varphi \Rightarrow \psi$), jestliže platí

$$\varphi = \varphi_1 A \varphi_2 , \quad \psi = \varphi_1 \alpha \varphi_2 ,$$

přičemž $A \to \alpha$ je jedno z přepisovacích pravidel. φ_1 nebo φ_2 může být také prázdný řetěz. Řekneme, že řetěz ψ je *generován* řetězem φ (znak $\varphi \overset{*}{\Rightarrow} \psi$), jestliže buďto $\psi = \varphi$, nebo existuje taková posloupnost řetězů

$$\varphi_1, \varphi_2, .., \varphi_n, n > 1 , \tag{1}$$

že platí

$$\varphi_1 = \varphi, \varphi_n = \psi, \varphi_i \Rightarrow \varphi_{i+1} \qquad \text{pro } 1 \leqq i < n .$$

Posloupnost (1) nazýváme *derivací* (řetězu φ_n z řetězu φ_1).

Příklad 1. Budiž $\{a, b, c\}$ terminální abeceda a buďtež $A \to c$, $A \to aAb$ přepisovací pravidla. Potom

$$cAcbaA , \quad caAbcbaA , \quad caAbcbac$$

je derivace.

Příklad 2. V syntaxi programovacího jazyka ALGOL 60 jsou metalingvistické proměnné označeny slovy, která do jisté míry vyznačují jejich význam, a která jsou uzavřena do úhlových závorek ⟨ ⟩ (viz [3]). Přepisovací pravidla jsou psána pomocí tzv. *Backusovy normální formy* (Bnf). To znamená, že všechna pravidla s touž levou stranou jsou shrnuta do jednoho a jednotlivé pravé strany jsou navzájem odděleny symbolem |. Místo znaku → se užívá znaku ::=. Uvažme pravidla

$$\begin{cases} \langle \text{číslice} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ \langle \text{celé číslo bez znaménka} \rangle ::= \langle \text{číslice} \rangle \mid \langle \text{celé číslo bez znaménka} \rangle \langle \text{číslice} \rangle \end{cases} \tag{2}$$

Tedy

$$\begin{aligned} &\langle \text{celé číslo bez znaménka} \rangle, \langle \text{celé číslo bez znaménka} \rangle \langle \text{číslice} \rangle , \\ &\langle \text{celé číslo bez znaménka} \rangle 4, \langle \text{celé číslo bez znaménka} \rangle \langle \text{číslice} \rangle 4 , \\ &\langle \text{číslice} \rangle \langle \text{číslice} \rangle 4, 0 \langle \text{číslice} \rangle 4, 074 \end{aligned} \tag{3}$$

je derivace, pomocí které se generuje z metaproměnné ⟨celé číslo bez znaménka⟩ desítkové číslo 074.

Gramatikou formálního jazyka nazýváme čtveřici (V_T, V_N, P, S), kde V_T je termi-

nální abeceda, V_N je množina metalingvistických proměnných, $\dot{P}$ množina přepisovacích pravidel a $S \in V_N$ je vytčená metaproměnná. *Jazykem* L vytvořeným gramatikou (V_T, V_N, P, S) se nazývá množina všech řetězů φ nad abecedou V_T takových, že platí $S \overset{*}{\Rightarrow} \varphi$ (při užití pravidel z P). Podle úmluvy výše učiněné nazýváme tyto řetězy frázemi jazyka L.

Příklad 3. Uvažme gramatiku $(\{a, b\}, \{A\}, \{A \to ab, A \to aAb\}, A)$. Jazyk vytvořený touto gramatikou je množina všech řetězů $a^n b^n$ pro $n \geqq 1$. Přitom např. a^n značí řetěz $a \ldots a$ vzniklý zřetězením n symbolů a.

Příklad 4. Nechť V_T je množina $\{0, 1\}$ (dvojkových číslic), $V_N = \{\langle$číslice$\rangle, \langle$celé číslo bez znaménka$\rangle\}$, P je tvořena pravidly (2) a $S = \langle$celé číslo bez znaménka$\rangle$. Jazyk vytvořený touto gramatikou je množina všech celých nezáporných dyadických čísel, tj. množina všech řetězů z nul a jedniček.

Je patrno, že obecně existuje více derivací realizujících generování řetězu z nějakého jiného řetězu. Je nám totiž ponecháno při každém kroku na libovůli, za kterou z metaproměnných, obsažených v daném nebo při derivaci vzniklém řetězu, použijeme příslušného přepisovacího pravidla. Je-li z nějakých důvodů potřeba vybrat některou z možných derivací, užívá se např. pojmů *nejlevější* nebo *nejpravější* derivace. Při první z nich je předepsáno dosazovat vždy za první metaproměnnou zleva; při druhé dosazujeme při každém bezprostředním generování za poslední metaproměnnou v řetězu. Tak např. každá derivace generující libovolný řetěz $a^n b^n$ z metaproměnné A v příkladě 3 je současně nejpravější i nejlevější derivací. Derivace (3) v příkladě 2 není ani nejlevější ani nejpravější.

Derivaci fráze jazyka z vytčené metaproměnné lze znázornit rovinným grafem, a to *ohodnoceným orientovaným stromem*. Ukážeme si sestrojení takového *derivačního stromu* na příkladě.

Příklad 5. Uvažme gramatiku s pravidly

1. $S \to AB$ 4. $B \to bc$
2. $A \to a$ 5. $B \to Bd$
3. $A \to ABb$

Terminální abeceda je tedy $\{a, b, c\}$, metaproměnné jsou A, B, S, přičemž S budiž vytčená metaproměnná. Zvolme si derivaci

$$S, \quad AB, \quad aB, \quad aBd, \quad abcd. \tag{4}$$

Grafické znázornění této (nejlevější) derivace se opírá o tyto zásady:

1. Každý vrchol derivačního stromu je ohodnocen jedním symbolem, a to buďto metaproměnnou nebo terminálním symbolem.

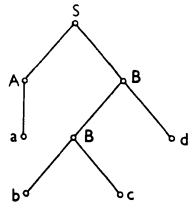
2. Každá (orientovaná) hrana vychází vždy z vrcholu ohodnoceného metaproměnnou a končí ve vrcholu ohodnoceném symbolem, který se nachází na pravé straně pravidla, pomocí kterého je tato metaproměnná v příslušném kroku derivace přepsána.

3. Ve 2 zmíněné přiřazení hran a symbolů z pravé strany příslušného pravidla je jednojednoznačné.

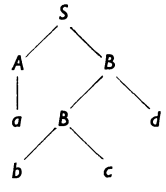
4. Kořen stromu je ohodnocen vytčenou metaproměnnou.

5. Pořadí hran vycházejících z jednoho vrcholu (uvažováno v kladném smyslu otáčení) odpovídá pořadí příslušných symbolů na pravé straně pravidla.

Tedy derivaci (4) znázorníme stromem z obr. 1.



Obr. 1.



Obr. 2.

Často se při kreslení derivačního stromu zakreslují znaky, jimiž jsou ohodnoceny jednotlivé vrcholy, na místa těchto vrcholů. Na obr. 2 je tímto způsobem vytisknut graf z obr. 1.

Je patrné, že derivační strom, sestrojený podle výše uvedených pravidel, není grafickým znázorněním pouze jedné vybrané derivace. Popisuje vůbec všechny derivace, které se navzájem liší jen pořadím dosazování za jednotlivé metaproměnné, obsažené v průběžných řetězcích, přičemž se vždy používá týchž přepisovacích pravidel. Říkáme, že derivační strom popisuje *syntaktickou strukturu* fráze (vzhledem k dané gramatice).

Snadno zjistíme, že v gramatice z příkladu 5 má fráze *abcd* pouze jedinou syntaktickou strukturu, tj. přísluší jí jediný derivační strom, a tedy jediná nejlevější (a také jediná nejpravější) derivace. Důkaz si může čtenář provést jako cvičení. Nemusí tomu být vždy takto. To ukáže následující příklad.

Příklad 6. Uvažme gramatiku s pravidly

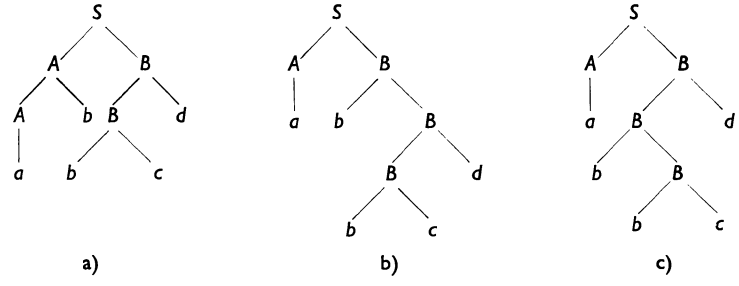
1. $S \to AB$ 4. $B \to bc$
2. $A \to a$ 5. $B \to bB$
3. $A \to Ab$ 6. $B \to Bd$

a frázi *abbcd*. Existují tři syntaktické struktury tohoto řetězu v dané gramatice. Jsou charakterizovány třemi různými derivačními stromy a), b), c) z obr. 3.

Dá se ukázat, že existuje $\frac{1}{2}n(n+1)$ různých syntaktických struktur fráze $ab^n cd$ v uvažované gramatice.

Fráze jazyka vytvořeného danou gramatikou se nazývá *jednoznačná* vzhledem k této gramatice, jestliže má v této gramatice pouze jedinou syntaktickou strukturu. Gramatika se nazývá *jednoznačná*, jestliže každá fráze jazyka jí vytvořeného je jednoznačná. Otázka jednoznačnosti gramatiky je *algoritmicky nerozhodnutelná.*

To znamená, že neexistuje algoritmus, který by pro libovolnou předloženou gramatiku rozhodl, zda je či není jednoznačná.



Obr. 3.

Vraťme se na chvilku k příkladu 3. Je patrno, že gramatika, jejíž pravidla (v Bnf) jsou

$$A \to ab \mid Bb \qquad B \to aA$$

vytváří (pro vytčenou metaproměnnou A) tentýž jazyk $\{a^n b^n : n \geqq 1\}$ jako gramatika z příkladu 3. Obecně existuje nekonečně mnoho různých gramatik, které vytvářejí stejný jazyk. Může se stát, že některá z těchto gramatik je jednoznačná a jiná ne. Vzniká otázka, zda existují *vnitřně nejednoznačné* jazyky, pro něž všechny gramatiky je vytvářející jsou nejednoznačné. Bylo dokázáno, že takové vnitřně nejednoznačné jazyky existují. Příkladem takového jazyka je množina

$$\{a^n b^m c^p : n = m \quad \text{nebo} \quad m = p\}$$

Tento příklad pochází od R. J. Parikha. Každá fráze $a^n b^n c^n$ tohoto jazyka má v každé gramatice alespoň dvě syntaktické struktury. Existuje gramatika, v níž mají všechny tyto fráze právě dvě struktury.

Naproti tomu existují na pohled jednoduché množiny řetězů nad konečnou abecedou, které nejsou bezkontextovými jazyky, tj. nedají se vytvořit žádnou gramatikou ve smyslu výše popsaném (*bezkontextovou gramatikou*). Je to např. množina

$$\{a^n b^n a^n : n \geqq 1\} \tag{5}$$

nad konečnou abecedou $\{a, b\}$. Tato množina je však tzv. *kontextovým jazykem*, tj. dá se vytvořit pomocí tzv. *kontextové gramatiky*. Pravidla takové gramatiky jsou tvaru

$$\alpha A \beta \to \alpha \gamma \beta ,$$

kde α, β jsou řetězy (případně prázdné) metalingvistických proměnných a γ je neprázdný řetěz z metaproměnných a terminálních symbolů. Množina (5) se dá vytvořit

kontextovou gramatikou tvořenou pravidly

$$\begin{array}{ll} S \to ABA & DE \to BE \\ AB \to AABBC & BE \to BC \\ CA \to AA & A \to a \\ CB \to DB & B \to b \\ DB \to DE & \end{array}$$

Vytčenou metaproměnnou je zde S. Čtenář nechť se sám přesvědčí, že postupným přepisováním řetězu ABA s použitím uvedených přepisovacích pravidel získá všechny fráze $a^n b^n a^n$, $n \geqq 1$ jakožto řetězy generované z S a jen tyto.

V dalším se budeme zabývat pouze bezkontextovými gramatikami a jazyky, a proto zde nebudeme uvádět již žádné další vlastnosti kontextových (a jiných obecnějších) gramatik.

SYNTAKTICKÁ ANALÝZA SHORA

Syntaktickou analýzou rozumíme proces získání (aspoň jedné) syntaktické struktury pro frázi bezkontextového jazyka, je-li dána příslušná (bezkontextová) gramatika. Algoritmy tohoto procesu jsou zpravidla sestaveny tak, aby pro libovolný konečný řetěz nad terminální abecedou uvažovaného jazyka rozhodly, zda je frází tohoto jazyka, a aby v kladném případě umožňovaly najít derivační strom. Nepožaduje se zpravidla, aby byly nalezeny všechny derivační stromy, jde-li o nejednoznačnou frázi. Algoritmy hledající všechny syntaktické struktury pro danou frázi mají obyčejně tu nepříjemnou vlastnost, že práce podle nich trvá dlouho. Při praktickém použití syntaktické analýzy se autoři kompilátorů spoléhají zpravidla na to, že zkoumaný jazyk je jednoznačný. Dokonce se v takových případech používá algoritmů, kterými se dají analyzovat pouze jazyky, jejichž gramatiky podléhají různým omezením. Uvidíme v dalším, jaká omezení to mohou být. Přesto si ve čtvrtém odstavci popíšeme proces syntaktické analýzy poskytující všechny derivační stromy dané fráze v dané gramatice.

Velmi názorný popis procesu syntaktické analýzy, který si nyní uvedeme, je převzat z článku [4]. Algoritmus analýzy využívá složité hierarchie podřízenosti jednotlivých metaproměnných. (Něco málo o této závislosti viz v [1], oddíl 3.) Užijeme metafory. Představme si, že nějaký člověk je pověřen úkolem analyzovat frázi bezkontextového jazyka, má-li k dispozici příslušnou gramatiku. Má právo najímat podřízené, přidělovat jim úkoly a propouštět je při neúspěchu. Tito podřízení mají obdobná práva. Je dodržována úmluva, že každému je řečeno pouze jednou: „*Pokuste se najít G*“, kde G je symbol jazyka (metalingvistická proměnná nebo terminální symbol) a může mu pak být opakovaně řečeno: „*Pokuste se znova*“, jestliže se jím nalezený řetěz ukázal jeho nadřízenému jako nevyhovující. V závislosti na G, speciálně na tvaru příslušné definice, každý pracovník (označme si ho na chvíli P) má použít vhodné strategie, kterou zformulujeme do následujících předpisů:

1. Je-li G terminální symbol a shoduje-li se s následujícím symbolem analyzované fráze, P posune ukazatele ve frázi o jedno místo dále (např. slabě tužkou přeškrtne příslušný symbol) a hlásí svému nadřízenému úspěch. Neshoduje-li se G s následujícím symbolem fráze, P hlásí neúspěch svému bezprostředně nadřízenému. Je-li mu po jeho úspěchu řečeno jeho nadřízeným, aby se pokusil znova, P posune ukazatele o jedno místo zpět (např. vymaže přeškrtnutí posledně přeškrtnutého symbolu) a hlásí neúspěch.

2. Jsou-li G metaproměnná a G_1 symbol (metaproměnná nebo terminální symbol) takové, že $G \to G_1$, P najme podřízeného P_1 příkazem: „*Pokuste se najít* G_1". P opakuje hlášení, které dostal od P_1, svému bezprostředně nadřízenému a propouští P_1, jestliže ten hlásí neúspěch. Je-li mu (P) řečeno, aby se pokusil znova (což může nastat pouze tehdy, jestliže měl před tím úspěch, který hlásil svému nadřízenému a který byl opakováním hlášení úspěchu, které dostal od P_1), P musí pověřit P_1 novým pokusem, znova sděluje hlášení, které dostal od P_1, svému nadřízenému a propouští P_1 při neúspěchu.

3. Jsou-li G metaproměnná a G_i, $i = 1, \ldots, n$, symboly (metaproměnné nebo terminální symboly) takové, že $G \to G_1 G_2 \ldots G_n$, P najímá postupně vždy jednoho podřízeného P_i pro každý G_i příkazem: „Pokuste se najít G_i". Jestliže P_i má úspěch, P zvětší i o jedničku, najme nového podřízeného a tento postup se opakuje, pokud není $i > n$, kdy P hlásí úspěch. Nemá-li P_i úspěch, je propuštěn, i je zmenšeno o jedničku, a je-li $i > 0$, nový P_i (předchůdce toho, který neměl úspěch) je pověřen pokusit se znova. Je-li $i = 0$, P hlásí neúspěch, když již vyčerpal všechny způsoby nalezení G. Jestliže po úspěchu je mu řečeno, aby se pokusil znova, položí $i = n$, pověří P_i, aby se pokusil znova a po hlášení od P_i pokračuje jako předtím.

4. Jsou-li G metaproměnná a G_i, $i = 1, \ldots, n$, symboly (metaproměnné nebo terminální symboly) takové, že $G \to G_1 \mid G_2 \mid \ldots \mid G_n$, P najímá postupně vždy jednoho podřízeného pro každé G_i příkazem: „Pokuste se najít G_i". Nemá-li P_i úspěch, je propuštěn, i je zvětšeno o jedničku, je najat nový podřízený a postup se opakuje, pokud není $i > n$, kdy P hlásí neúspěch. Má-li P_i úspěch, P hlásí úspěch. Je-li P po úspěchu pověřen pokusit se znova, řekne P_i (tomu, který měl úspěch), aby se pokusil znova, a po hlášení od P_i pokračuje jako předtím.

5. Všechny složitější definice mohou být považovány za sestavené z předcházejících čtyř typů.

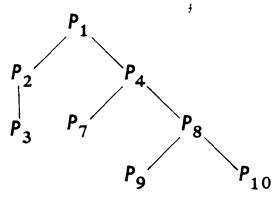
Ukažme si použití uvedeného postupu při analýze fráze *abbcd* pomocí gramatiky z příkladu 6. Zapíšeme si ji zde pomocí Bnf, aby bylo průhlednější použití bodu 4 výše zformulovaných předpisů.

$$S \to AB \qquad A \to a \mid AB \qquad B \to bc \mid bB \mid Bd$$

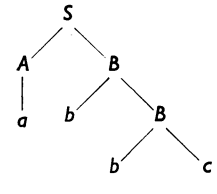
Analýza bude probíhat nyní takto: P_1 je pověřen úkolem najít S; najme P_2 příkazem najít A; P_2 najme P_3 příkazem najít a (podle prvního pravidla s levou stranou A); P_3 přeškrtne a ve frázi a hlásí úspěch P_2; P_2 hlásí úspěch P_1; P_1 najme P_4 příkazem

najít B; P_4 najme P_5 příkazem najít b (podle prvního pravidla s levou stranou B); P_5 škrtne b (druhý symbol ve frázi) a hlásí úspěch P_4; P_4 najme P_6 příkazem najít c (stále podle prvního pravidla pro B); P_6 hlásí neúspěch P_4; P_4 propustí P_6 a pověří P_5 novým pokusem; P_5 odkryje b (druhý symbol ve frázi) a hlásí neúspěch P_4; P_4 propustí P_5 a najme P_7 příkazem najít b (podle druhého pravidla pro B); P_7 škrtne b (druhý symbol ve frázi) a hlásí úspěch P_4; P_4 najme P_8 příkazem najít B (stále podle druhého pravidla pro B); P_8 najme P_9 příkazem najít b (podle prvního pravidla pro B); P_9 škrtne b (třetí symbol ve frázi) a hlásí úspěch P_8; P_8 najme P_{10} příkazem najít c (stále podle prvního pravidla pro B); P_{10} škrtne c (čtvrtý symbol ve frázi) a hlásí úspěch P_8; P_8 hlásí úspěch P_4 (neboť skončilo použití celé pravé strany prvního pravidla pro B); P_4 hlásí úspěch P_1, neboť skončilo použití celé pravé strany druhého pravidla pro B (jehož vyzkoušením byl P_4 pověřen).

Podívejme se nyní pomocí grafu podřízenosti (obr. 4), co jsme dostali. Nahradíme-li



Obr. 4.



Obr. 5.

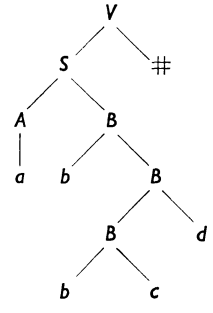
v tomto grafu jména pracovníků označením úkolů, které dostali, dostaneme derivační strom z obr. 5. Ten popisuje syntaktickou strukturu fráze *abbc*, která sice též patří do uvažovaného jazyka, ale není to předložená fráze. Vidíme, že ve výše uvedeném (metaforickém) popisu procedury pro syntaktickou analýzu chybí kontrola toho, zda v okamžiku, kdy první pověřený pracovník (P_1) by měl hlásit úspěch, jsme už vyčerpali celou předloženou frázi.

Tento požadavek může být snadno splněn malou modifikací gramatiky. Před první přepisovací pravidlo pro vytčenou metaproměnnou (třeba S) umístíme pravidlo $V \rightarrow S \#$, kde $\#$ je symbol, který se liší od všech symbolů dosavadní terminální abecedy. Kromě toho připojíme symbol $\#$ vždy nakonec každé fráze, předložené k analýze. První pracovník je tedy nyní vždy pověřen nalezením V. Je patrno, že ke hlášení úspěchu pracovníkem P_1 může dojít nyní jedině až po prověření celé předložené fráze. Čtenář nechť si sám zjistí, že po právě popsané úpravě gramatiky dostane s použitím výše popsaného postupu derivační strom z obr. 3b), ovšem ve tvaru z obr. 6.

Popsaný algoritmus poskytuje tedy pro každou frázi jazyka popis právě jedné syntaktické struktury. Případ, kdy předložený řetěz není frází jazyka, se projeví tím, že první pověřený pracovník by měl hlásit neúspěch.

Na první pohled by se mohlo zdát, že k dalším dvěma syntaktickým strukturám fráze *abbcd* v gramatice z příkladu 6 bychom mohli při použití výše popsaného postupu dospět změnou pořadí pravidel v gramatice. Tu však narážíme na obtíž, která ve

svých důsledcích vede k jednomu druhu omezení pro gramatiky při některých způsobech syntaktické analýzy. Kdybychom v uvažované gramatice např. vyměnili mezi sebou pravidla 1 a 2 (s úmyslem získat syntaktickou strukturu odpovídající případu a) na obr. 3), zauzlil by se proces analýzy v nekonečněkrát opakovaný cyklus najímání stále nových a nových pracovníků k nalezení A podle (nyní) prvního pravidla



Obr. 6.

$A \to Ab$ s levou stranou A. Musí se tedy při analýzách prováděných zleva (mezi něž patří též uvedený způsob) buďto požadovat, aby gramatika takováto, tzv. *zleva rekurzívní* pravidla neobsahovala, nebo jim musí v pořadí předcházet pravidlo se stejnou levou stranou, ale nikoliv zleva rekurzívní. Obdobný problém vzniká také v případě, kdy nemáme sice pravidlo zleva rekurzívní, ale několik metaproměnných $A_1, \ldots, A_p$ s tou vlastností, že existují pravidla, která mají tvar

$$(6) \qquad A_1 \to A_2 \ldots, A_2 \to A_3 \ldots, \ldots, A_{p-1} \to A_p \ldots, A_p \to A_1 \ldots .$$

Jiný způsob, jakým je možno při analýze čelit vzniku zauzlení, je kontrola na délku řetězu, který vzniká pomocí dosud získané části derivačního stromu. Je-li již jeho délka větší nežli délka analyzované fráze, opakování se zastaví. Tento způsob kontroly však nestačí v případě, kdy tvar pravidel v (6) je

$$A_1 \to A_2, A_2 \to A_3, \ldots, A_{p-1} \to A_p, A_p \to A_1 ,$$

tj. jde vesměs o pravidla s jednočlennou pravou stranou. Nemůžeme se však zde zabývat těmito problémy.

Čtenář se může přesvědčit, že při všech „dovolených" pořadích pravidel 4, 5, 6 gramatiky z příkladu 6 (totiž 4, 5, 6; 4, 6, 5; 5, 4, 6; 5, 6, 4) dojdeme výše uvedeným postupem vždy k diagramu b) z obr. 3. Předpokládáme přitom samozřejmě, že bylo provedeno výše popsané doplnění pomocí symbolu #.

Nyní zbývá odůvodnit název analýza shora. Pochází od toho, že vrcholy derivačního stromu získáváme postupně v pořadí jejich podřízenosti, tj. od nadřízených k podřízeným.

Přepisovací pravidla se dají při analyzování řetězů použít též poněkud jiným způsobem, než bylo popsáno v předcházejícím odstavci. Naznačíme si nejdříve stručně princip druhého způsobu. Vyjdeme od předloženého řetězu a konstruujeme posloupnost řetězů tak, že každý z nich, počínaje druhým, vznikl z předchozího přepsáním nějaké jeho části pomocí některého z přepisovacích pravidel. Pravidel se přitom používá k přepisování v opačném smyslu nežli při syntéze frází. V každém z průběžných řetězů hledáme nějaký podřetěz, který je shodný s pravou stranou některého pravidla. Objevíme-li takovýto výskyt pravé strany, zaměníme tento podřetěz levou stranou příslušného pravidla, tj. metaproměnnou. Zbývající části řetězu ponecháme beze změny. Jestliže se nám takto podaří zkonstruovat posloupnost řetězů takovou, že její první člen je k analýze předložený řetěz a její poslední člen je vytčená metaproměnná, pak jsme zřejmě v opačném pořádku realizovali jednu z možných derivací vedoucích k dané frázi.

Je patrno, že některým z provedených přepsání může být nastoupena falešná cesta, tj. dospějeme třeba k řetězu, který se v žádné derivaci předložené fráze nemůže vyskytnout. Chyba, která se takto při analýze stala, může být ovšem objevena až v některém pozdějším stadiu, když se náhle ukáže, že nelze na získaný řetěz použít žádného z pravidel (tj. žádný jeho podřetěz se neshoduje s pravou stranou nějakého pravidla). Pak je nutno vrátit se zpět a zkoušet jiné možnosti přepisování. Jestliže jsme přes vyčerpání všech možností nedošli takto k vytčené metaproměnné, není předložený řetěz frází uvažovaného jazyka.

Naznačený postup se nazývá *analýzou zdola*, protože zde konstruujeme derivační strom tak, že od výsledné fráze, jejíž jednotlivé symboly jsou koncové vrcholy grafu (z nichž nevychází již žádná hrana), směřujeme k vrcholu popsanému vytčenou metaproměnnou, v němž naopak žádná hrana nekončí.

Aby se z popsaného postupu stal algoritmus, bylo by nutno předepsat pořadí, v jakém zkoušíme pravé strany jednotlivých pravidel, kolikátý případný její výskyt přepíšeme atd. Zformulujeme si tentokrát postup analýzy tak, abychom podle něho získali všechny syntaktické struktury analyzované fráze. Použijeme k tomu myšleného stroje, automatu, který je jedním druhem *Turingova stroje* se dvěma páskami. Tento způsob byl použit v článku [5].

Předpokládejme, že máme k dispozici automat, který používá dvou pásek k zápisu. Každou z nich předpokládáme v jednom směru (a to doleva) neomezenou. Na tyto pásky, které označíme T_1 a T_2, může automat zapisovat symboly terminální i neterminální abecedy (tj. metaproměnné) a symbol # a opět svůj zápis vymazávat. Symboly se zapisují na obě pásky těsně za sebou (bez mezer), počínaje od pravého konce. Řídíme se přitom instrukcemi, které budeme zapisovat ve tvaru

$$(7) \qquad (A_1 \dots A_m, C_1 \dots C_p) \to (B_1 \dots B_n, D_1 \dots D_q) .$$

Instrukce (7) říká, že je-li m, resp. p posledně zapsaných symbolů na pásce T_1, resp. T_2 rovno symbolům $A_1, \ldots, A_m$, resp. $C_1, \ldots, C_p$, má automat těchto m, resp. p symbolů nahradit n, resp. q symboly $B_1, \ldots, B_n$, resp. $D_1, \ldots, D_q$. Jinak řečeno, je-li (zleva) počáteční úsek řetězu napsaného na pásce T_1, resp. T_2 roven $A_1 \ldots A_m$, resp. $C_1 \ldots C_p$, nahradíme tento počáteční úsek řetězem $B_1 \ldots B_n$, resp. $D_1 \ldots D_q$. Abychom poznali, že jsme už dospěli na konec pásky, předpokládáme, že nejpravější symbol na obou páskách je vždy #. Objeví-li se nám v procesu jako první zleva, víme, že jsme na konci pásky.

Na začátku procesu předpokládáme vždy, že na pásce T_1 je zapsán řetěz předložený k analýze (a zakončený znakem #) a že na pásce T_2 je zapsána vytčená metaproměnná (a za ní ovšem opět symbol #). Jestliže na základě předepsaných instrukcí automat po konečnému počtu kroků (tj. přepisů na obou páskách) dospěje do stavu, v němž je na obou páskách zapsán pouze symbol #, řekneme, že analyzoval předloženou frázi. Z průběhu procesu je pak možno, jak hned uvidíme, získat popis její syntaktické struktury. Přitom nepožadujeme, aby automat byl deterministický. *Nemusí pro každý stav charakterizovaný příslušnou konfigurací, tj. současným obsahem obou pásek, existovat pouze nejvýše jeden následující stav, tj. nemusí se dát aplikovat vždy pouze jedna instrukce.* Dá-li se aplikovat např. k instrukcí, předpokládáme, že *automat od toho okamžiku sleduje všech k možných cest současně, paralelně.* Cesta končí, jestliže se na získanou konfiguraci nedá aplikovat žádná instrukce. Automat se zastaví, jsou-li ukončeny všechny nastoupené cesty.

Popíšeme si nyní, jak na základě gramatiky získáme potřebné instrukce. Je-li S vytčená metaproměnná, máme především instrukci

$$(S, S) \to (\Lambda, \Lambda), \tag{8}$$

kde Λ zde i v dalším značí prázdný řetěz. Je-li

$$A \to A_1 \ldots A_n \tag{9}$$

přepisovací pravidlo gramatiky, sestavíme na jeho základě instrukci

$$(A_n, A_{n-1} \ldots A_1) \to (A, \Lambda). \tag{10}$$

Konečně pro libovolný symbol B, terminální i neterminální (nikoliv #), budeme mít instrukci

$$(B, \Lambda) \to (\Lambda, B). \tag{11}$$

K významu těchto tří druhů instrukcí poznamenejme, že pomocí instrukcí (11) přepisujeme postupně první symboly z pásky T_1 na první volné místo na pásce T_2. Jakmile (třeba tímto způsobem) dospějeme k situaci, kdy první symbol na T_1 je posledním symbolem pravé strany nějakého pravidla, zatímco na T_2 máme na začátku napsánu obráceně zbývající část této pravé strany, můžeme též použít instrukce typu (10). Její použití znamená vlastně ve svém důsledku nahrazení výskytu pravé strany

pravidla v získaném řetězu levou stranou tohoto pravidla, tj. příslušnou metaproměnnou. Tato metaproměnná je potom zapsána jako první symbol na T_1. Vzniklá konfigurace umožňuje po několikerém případném použití instrukcí typu (11) opětné přepsání na základě jiného pravidla gramatiky atd. Použití instrukce (8) má za následek vymazání řetězu, který se ukázal být frází jazyka, tj. hodnotou vytčené metaproměnné S. Této instrukce se např. použije v posledním kroku správné cesty, kdy ponechává pak na obou páskách pouze symbol #.

Ukažme si teď průběh procesu opět na příkladě fráze *abbcd* v jazyku z příkladu 6. Budeme tedy ze začátku mít na pásce T_1 zapsán řetěz *abbcd* # a na pásce T_2 bude S #. Jednotlivé instrukce budou:

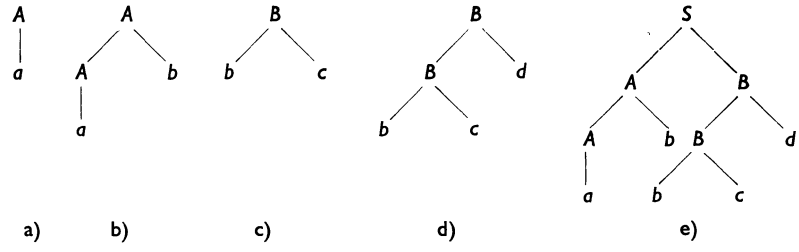
1. $(S, S) \rightarrow (\Lambda, \Lambda)$
2. $(B, A) \rightarrow (S, \Lambda)$
3. $(a, \Lambda) \rightarrow (A, \Lambda)$
4. $(b, A) \rightarrow (A, \Lambda)$
5. $(c, b) \rightarrow (B, \Lambda)$
6. $(B, b) \rightarrow (B, \Lambda)$
7. $(d, B) \rightarrow (B, \Lambda)$
8. $(S, \Lambda) \rightarrow (\Lambda, S)$
9. $(A, \Lambda) \rightarrow (\Lambda, A)$
10. $(B, \Lambda) \rightarrow (\Lambda, B)$
11. $(a, \Lambda) \rightarrow (\Lambda, a)$
12. $(b, \Lambda) \rightarrow (\Lambda, b)$
13. $(c, \Lambda) \rightarrow (\Lambda, c)$
14. $(d, \Lambda) \rightarrow (\Lambda, d)$.

Vyznačme si nyní tři cesty, které vedou k popisu tří syntaktických struktur uvažovaného řetězu, posloupnostmi příslušných konfigurací, tj. současných obsahů obou pásek. Před každou konfigurací (s výjimkou vstupní, počáteční) napišme číslo, které udává instrukci, pomocí které byla tato konfigurace získána z předchozí. Nebudeme zde vyznačovat falešné cesty, které popisují nesprávný průběh analýzy, tj. vedoucí ke konfiguracím různým od # # a takovým, že se na ně nedá již použít žádné instrukce.

Z průběhů procesu se dá nyní odvodit konstrukce derivačního stromu, a to, jak uvidíme, zdola. Ukažme si to třeba na průběhu 1. Použití instrukce 3 znamená, že

instrukce	1.		instrukce	2.		instrukce	3.	
	páska T_1	páska T_2		páska T_1	páska T_2		páska T_1	páska T_2
	abbcd #	S #		*abbcd* #	S #		*abbcd* #	S #
3	*Abbcd* #	S #	3	*Abbcd* #	S #	3	*Abbcd* #	S #
9	*bbcd* #	AS #	9	*bbcd* #	AS #	9	*bbcd* #	AS #
4	*Abcd* #	S #	12	*bcd* #	bAS #	12	*bcd* #	bAS #
9	*bcd* #	AS #	5	*Bd* #	bAS #	12	*cd* #	$bbAS$ #
12	*cd* #	bAS #	6	*Bd* #	AS #	5	*Bd* #	bAS #
5	*Bd* #	AS #	10	*d* #	BAS #	10	*d* #	$BbAS$ #
10	*d* #	BAS #	7	*B* #	AS #	7	*B* #	bAS #
7	*B* #	AS #	2	S #	S #	6	*B* #	AS #
2	S #	S #	1	#	#	2	S #	S #
1	#	#				1	#	#

při derivaci bylo použito pravidlo $A \to a$. Získáváme tak část stromu vyznačenou na obr. 7a). Následující použití instrukce 9 je přípravou k použití instrukce 4. Oba



Obr. 7.

kroky dohromady říkají, že A, získané na obr. 7a), je prvním a b druhým symbolem pravé strany pravidla, které bylo v derivaci použito před tím, tj. pravidla $A \to Ab$. Dostáváme tak část stromu vyznačenou na obr. 7b). Následující instrukce 9 a 12 jsou přípravou na použití instrukce 5. Dohromady poslední dva kroky znamenají, že bylo při derivaci použito pravidla $B \to bc$, což vede k té části derivačního stromu vyznačené na obr. 7c). Metaproměnná A (odpovídající nejvyššímu A na obr. 7b)) je zatím uschována na pásce T_2 k pozdějšímu (při derivaci vlastně dřívějšímu) použití. Následující použití instrukce 10 je přípravou k použití instrukce 7. Ta říká, že při derivaci byla metaproměnná B z obr. 7c) získána jako první symbol pravé strany použitého pravidla $B \to Bd$. Dostáváme tedy část stromu vyznačenou na obr. 7d). Konečně použití instrukce 2 konstatuje použití pravidla $S \to AB$, což spojuje části z obr. 7b) a 7d) do výsledného stromu na obr. 7e). Příznivý výsledek instrukce 1 říká že jednak byla dokončena analýza celého řetězu, jednak že tento řetěz je frází uvažovaného jazyka. Poskytla nám tedy cesta 1 derivační strom z obr. 3a). Čtenář nechť se laskavě přesvědčí, že zbývající dvě cesty 2 a 3 vedou ke stromům z obr. 3b) a 3c).

Existují způsoby, pomocí kterých lze omezit počet nastoupení falešných cest, a to jak pro analýzu zdola, tak pro analýzu shora. Jeden z těchto způsobů selekce se opírá o předem sestavenou booleovskou matici $P[X, Y]$. Každé metaproměnné odpovídá jeden řádek a každé metaproměnné i základnímu symbolu jeden sloupec matice. Položíme $P[X, Y]$ = ANO (tj. prvek matice na průsečíku řádku příslušného metaproměnné Y a sloupce příslušného prvku X je ANO), jestliže existuje řetěz, který se dá generovat z Y a začíná prvkem X. V opačném případě je $P[X, Y]$ = NE. Jinak řečeno, je $P[X, Y]$ = ANO právě tehdy, jestliže buďto existuje v gramatice přepisovací pravidlo s levou stranou Y, jehož pravá strana začíná prvkem X, nebo existuje pravidlo, jehož pravá strana začíná prvkem X a jehož levá strana je prvním symbolem pravé strany pravidla s levou stranou Y, atd. Využijeme-li této booleovské matice na vhodném místě popisu procedury syntaktické analýzy, dají se tím eliminovat některé falešné cesty analýzy již před jejich nastoupením. Nebudeme zde však tento problém dále rozebírat.

SYNTAKTICKÁ ANALÝZA SE SOUČASNOU SYNTÉZOU VÝSTUPNÍHO TEXTU

Edgar T. IRONS sestavil proceduru, pomocí které lze současně s jednotlivými kroky, z nichž se sestává syntaktická analýza bezkontextového jazyka, vytvářet části překladu daného vstupního textu. Syntéza výstupního textu je dokončena současně s dokončením analýzy vstupního textu. I když tato procedura, nazvaná autorem DIAGRAM a popsaná např. v práci [6], má některé nedostatky, můžeme si zde říci několik slov o principech postupů, kterých využívá. Nejlépe to znázorníme opět na příkladu.

Vyjdeme z části gramatiky (syntaxe) mezinárodního programovacího jazyka ALGOL 60 (viz [3]). Uvážíme tu její část, která popisuje sestavování *aritmetických výrazů*. Neprovedeme to ovšem zde v celé úplnosti a užijeme poněkud změněného označení. Mějme tedy tato pravidla psaná v Bnf:

(12)
$$
\begin{cases}
\langle\text{písm}\rangle \to a \mid b \mid c \mid \ldots \mid z \\
\langle\text{čísl}\rangle \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
\langle\text{iden}\rangle \to \langle\text{písm}\rangle \mid \langle\text{iden}\rangle\,\langle\text{písm}\rangle \mid \langle\text{iden}\rangle\,\langle\text{čísl}\rangle \\
\langle\text{prv výr}\rangle \to \langle\text{iden}\rangle \mid (\langle\text{aritm výr}\rangle) \\
\langle\text{op typu nás}\rangle \to \times \mid / \\
\langle\text{op typu sčít}\rangle \to + \mid - \\
\langle\text{člen}\rangle \to \langle\text{prv výr}\rangle \mid \langle\text{člen}\rangle\,\langle\text{op typu nás}\rangle\,\langle\text{prv výr}\rangle \\
\langle\text{souč}\rangle \to \langle\text{člen}\rangle \mid \langle\text{souč}\rangle\,\langle\text{op typu sčít}\rangle\,\langle\text{člen}\rangle \\
\langle\text{aritm výr}\rangle \to \langle\text{souč}\rangle
\end{cases}
$$

Význam pravidel pro *písmena* a *číslice* (první dva řádky v (12)) je jasný. Pravidla pro ⟨iden⟩, tj. pravidla pro vytváření *identifikátorů* (tj. označení) *proměnných* nám říkají, že identifikátor je libovolná posloupnost písmen a číslic, ale začínající písmenem (viz též [1]). Následující pravidla pro *prvotní výraz* ukazují, že za prvotní výraz považujeme libovolný identifikátor, ale také libovolnou posloupnost znaků, která se dá podle naší gramatiky rozpoznat jako aritmetický výraz, ale uzavřenou do kulatých závorek. Další pravidla pro *operátor typu násobení* a *operátor typu sčítání* jsou opět jasná. *Člen* je pak buďto samotný prvotní výraz, nebo (už dříve získaný) člen spojený pomocí operátoru typu násobení s nějakým prvotním výrazem. *Součet* (předposlední řádek pravidel) je buďto libovolný člen, nebo (už dříve získaný) součet spojený pomocí operátoru typu sčítání s nějakým členem. Konečně za *aritmetický výraz* považujeme libovolný součet.

Ukažme si několik příkladů:

⟨iden⟩ : a, $c13$, *omega*, *vrchol* 16
⟨prv výr⟩ : $ad105\ c4$, $(b1 + b2)$, $(a - (b1 + b2)/c)$
⟨člen⟩ : $(b1 - b2)$, $(b1 + b2)/c$
⟨souč⟩ : $c \times (b1 - b2)$, $bc + (a1 - a2/b) \times (b1 + b2)$, $a1 \times a2 + (b2 - b1)/max$

Příklady pro součet jsou současně příklady pro aritmetický výraz.

Pro použití v proceduře DIAGRAM jsou přepisovací pravidla psána s navzájem vyměněnými pravými a levými stranami, aby bylo vyznačeno jejich použití pro analýzu zdola, kdy přepisujeme výskyt původní pravé strany na metaproměnnou z původní levé strany. Kromě toho je za každé takto modifikované pravidlo připojen ve složených závorkách popis jeho významu ve výstupním jazyku. Má tedy nyní každé pravidlo tvar

$$A_1 \dots A_n \rightarrow A\{V_1 \dots V_m\} \tag{13}$$

Každý ze symbolů V_i může být buďto terminální symbol (v) výstupního jazyka, nebo výraz tvaru

$$\varrho_j[v \leftarrow v_1 \dots v_p]\,, \tag{14}$$

kde hranaté závorky i s obsahem mohou chybět. Obdobně jako v předcházejícím odstavci provádíme analýzu zdola tím, že hledáme výskyt levých stran pravidel tvaru (13) v postupně modifikovaném vstupním řetězu a za nalezený výskyt dosadíme pravou stranu příslušného pravidla. Nebudeme se nyní starat o to, jakým způsobem je zformulován algoritmus, tj. jak se vracíme z případně nastoupených falešných cest apod. Zajímá nás hlavně ta okolnost, že při každém správném přepsání vstupního řetězu pomocí pravidla (13) zapíšeme část výstupního textu. Ta vznikne z výrazu obsaženého ve složených závorkách v pravidle (13) tak, že

a) jednotlivé terminální symboly v výstupního jazyka ponecháme na místě a

b) symbol V_i rovný výrazu (14) nahradíme řetězem, který jsme dosud získali pro j-tý symbol levé strany pravidla (13) počítáno od znaménka $\rightarrow$ vlevo, přičemž za každý výskyt symbolu v v něm dosadíme řetěz $v_1 \dots v_p$.

Uvedeme si nejdříve zcela jednoduchý příklad. Uvažme část gramatiky (12), a to

⟨písm⟩ → $a \mid b$
⟨iden⟩ → ⟨písm⟩ | ⟨iden⟩ ⟨písm⟩

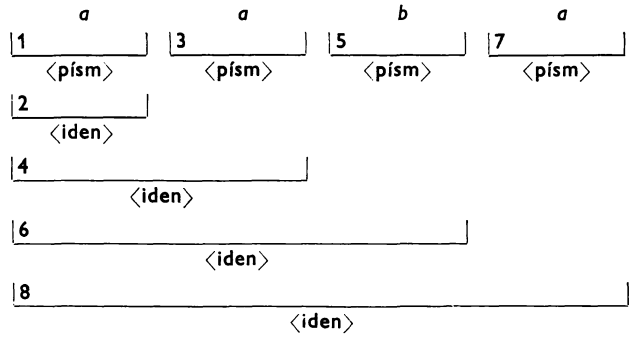
Z těchto pravidel zkonstruujeme pravidla tvaru (13), a to

(15.1) $a \rightarrow$ ⟨písm⟩ $\{A \sim TI\}$
(15.2) $b \rightarrow$ ⟨písm⟩ $\{B \sim TI\}$
(15.3) ⟨písm⟩ → ⟨iden⟩ $\{\varrho_1\}$
(15.4) ⟨iden⟩ ⟨písm⟩ → ⟨iden⟩ $\{\varrho_2[T \leftarrow TI], \varrho_1\}$

Tedy A, B, T, I, $\sim$ jsou terminální symboly výstupního jazyka. Proveďme si analýzu identifikátoru *aaba* a vyznačme si její výsledek schématem na obr. 8.

Číslice v rozích jednotlivých ležatých hranatých závorek udávají, v kolikátém kroku analýzy byla získána ta metaproměnná, jejíž symbol je zapsán pod touto závorkou. Vidíme, že v prvním kroku byl přepsán terminální symbol a vstupního jazyka na metaproměnnou ⟨písm⟩ užitím pravidla (15.1). Takto získaná metaproměnná byla ve druhém kroku přepsána na metaproměnnou ⟨iden⟩ užitím pravidla (15.3). Ve třetím kroku byl druhý symbol a vstupního řetězu přepsán na metaproměnnou ⟨písm⟩

opět užitím pravidla (15.1). Nyní užijeme pravidla (15.4) a přepíšeme získaný řetěz ⟨iden⟩ ⟨písm⟩ na ⟨iden⟩ atd. Následnost jednotlivých kroků není určena jednoznačně, např. je možno vyměnit pořadí kroků 4 a 5 apod. To nás však nyní nezajímá.



Obr. 8.

Zapišme nyní, co dostaneme při jednotlivých krocích na výstupu (čísla označující kroky jsou v levém sloupci, části výstupního řetězu v pravém):

1, 2 $A \sim TI$
3 $A \sim TI$
4 $A \sim TII, A \sim TI$
5 $B \sim TI$
6 $A \sim TIII, A \sim TII, B \sim TI$
7 $A \sim TI$
8 $A \sim TIII, A \sim TIII, B \sim TII, A \sim TI$

V posledním řádku máme ve druhém sloupci „překlad" vstupního řetězu do výstupního jazyka. Můžeme ho chápat třeba tak, že počet velkých *I* za *T* udává, kolikátému symbolu zprava ve vstupním řetězu odpovídá příslušné velké písmeno před znakem $\sim$.

Předpokládejme nyní, že máme jednoadresový samočinný počítač, jehož některé instrukce jsou popsány v následující tabulce (viz např. [7]).

Kód instrukce	Význam
ZÁP — *m*	zápis obsahu paměťové buňky *m* do střadače
SČT — *m*	obsah buňky *m* přičti k obsahu střadače
ODČ — *m*	obsah buňky *m* odečti od obsahu střadače
NÁS — *m*	obsah buňky *m* násob obsahem střadače
DĚL — *m*	obsah střadače děl obsahem buňky *m*
PŘN — *m*	obsah střadače přemísti do buňky *m*

Přepišme si nyní pravidla (12) uvažované gramatiky na tvar (13) tak, abychom po syntaktické analýze aritmetického výrazu na vstupu dostali na výstupu program pro výpočet jeho hodnoty na uvažovaném samočinném počítači. Bude to ovšem tzv. *program v symbolických adresách*, tj. příslušná buňka paměti bude označena identifikátorem té proměnné, pro záznam jejíchž hodnot je rezervována. Pro pomocné buňky určené k dočasnému uložení průběžných výsledků volíme označení *T*, *TI*, atd. Dostáváme pravidla

$a \to \langle \text{písm} \rangle \{A\}$
$b \to \langle \text{písm} \rangle \{B\}$
$\vdots$
$z \to \langle \text{písm} \rangle \{Z\}$
$0 \to \langle \text{čísl} \rangle \{0\}$
$1 \to \langle \text{čísl} \rangle \{1\}$
$\vdots$
$9 \to \langle \text{čísl} \rangle \{9\}$
$\langle \text{písm} \rangle \to \langle \text{iden} \rangle \{\varrho_1\}$
$\langle \text{iden} \rangle \langle \text{písm} \rangle \to \langle \text{iden} \rangle \{\varrho_2\varrho_1\}$
$\langle \text{iden} \rangle \langle \text{čísl} \rangle \to \langle \text{iden} \rangle \{\varrho_2\varrho_1\}$
$\times \to \langle \text{op typu nás} \rangle \{\text{NÁS}\}$
$/ \to \langle \text{op typu nás} \rangle \{\text{DĚL}\}$
$+ \to \langle \text{op typu sčít} \rangle \{\text{SČT}\}$
$- \to \langle \text{op typu sčít} \rangle \{\text{ODČ}\}$
$\langle \text{iden} \rangle \to \langle \text{prv výr} \rangle \{\text{ZÁP} - \varrho_1\}$
$(\langle \text{aritm výr} \rangle) \to \langle \text{prv výr} \rangle \{\varrho_2\}$
$\langle \text{prv výr} \rangle \to \langle \text{člen} \rangle \{\varrho_1\}$
$\langle \text{člen} \rangle \langle \text{op typu nás} \rangle \langle \text{prv výr} \rangle \to \langle \text{člen} \rangle \{\varrho_1; \text{PŘN} - T; \varrho_3[T \leftarrow TI]; \varrho_2 - T\}$
$\langle \text{člen} \rangle \to \langle \text{souč} \rangle \{\varrho_1\}$
$\langle \text{souč} \rangle \langle \text{op typu sčít} \rangle \langle \text{člen} \rangle \to \langle \text{souč} \rangle \{\varrho_1; \text{PŘN} - T; \varrho_3[T \leftarrow TI]; \varrho_2 - T\}$
$\langle \text{souč} \rangle \to \langle \text{aritm výr} \rangle \{\varrho_1\}$

Zvolme si nyní ke zpracování poslední z příkladů na str. 279, tj. aritmetický výraz $a1 \times a2 + (b2 - b1)/max$. Schéma syntaktické analýzy je na tabulce na str. 286 Z něho je patrno postupné užití jednotlivých výše uvedených pravidel. Následuje záznam postupného vytváření programu při jednotlivých krocích analýzy (str. 283).

Průběh a zdárné zakončení procesu nám jednak potvrdilo, že daný řetěz je správně ve smyslu naší gramatiky sestaveným aritmetickým výrazem, jednak poskytlo program pro výpočet tohoto aritmetického výrazu na samočinném číslicovém počítači. Program byl hotov již při 43. kroku; 44. krok je nutný k tomu, abychom poznali, že jde o aritmetický výraz. Čtenář se snadno přesvědčí, že počítač vykoná na základě zkonstruovaného programu to, co se na něm požaduje. Výsledek výpočtu je nakonec uložen ve střadači.

Sestavený program není ovšem příliš efektivní. Využijeme-li hlubších vlastností aritmetických operací (např. komutativity sčítání a násobení) a provedeme-li některé další změny v gramatice (např. takovou, že samotný identifikátor nemůže být součtem

apod.), můžeme napsat pravidla v takovém tvaru, že získaný program bude mnohem efektivnější. Zamezí se např. zbytečná zasílání do střadače s bezprostředně následujícími přenosy do paměti. Tím se však zde nebudeme zabývat.

1,2	*A*	27	*B*1
3	1	28, 29	ZÁP – *B*1
4	*A*1	30, 31, 32, 33	ZÁP – *B*1; PŘN – *T*; ZÁP – *B*2; ODČ – *T*
5,6	ZÁP – *A*1	34	DĚL
7	NÁS	35, 36	*M*
8, 9	*A*	37	*A*
10	2	38	*MA*
11	*A*2	39	*X*
12	ZÁP – *A*2	40	*MAX*
13, 14	ZÁP – *A*2; PŘN – *T*; ZÁP – *A*1; NÁS – *T*	41	ZÁP – *MAX*
15	SČT	42	ZÁP – *MAX*; PŘN – *T*; ZÁP – *B*1; PŘN – *TI*; ZÁP – *B*2; ODČ – *TI*; DĚL – *T*
16, 17	*B*		
18	2		
19	*B*2		
20, 21, 22	ZÁP – *B*2	43, 44	ZÁP – *MAX*; PŘN – *T*; ZÁP – *B*1; PŘN – *TI*; ZÁP – *B*2; ODČ – *TI*; DĚL – *T*; PŘN – *T*; ZÁP – *A*2; PŘN – *TI*; ZÁP – *A*1; NÁS – *TI*; SČT – *T*
23	ODČ		
24, 25	*B*		
26	1		

ANALÝZA A KONTROLA TEXTU VYUŽÍVAJÍCÍ PŘEPISOVACÍCH PRAVIDEL NEPŘÍMO

Uvedeme krátce ještě jeden způsob prověřování textu, který je často užíván v praxi při kompilaci částí nebo celých programů psaných ve formálním programovacím jazyku. Nazývá se někdy *metodou přechodové tabulky* a průběh příslušného procesu se dá opět uvést v blízkou souvislost s prací Turingova automatu. Je popsán např. v [8]. Základním principem je využití okolnosti, že část informace, získávané při procházení vstupního textu symbol za symbolem, může být v každém okamžiku vtělena do hodnoty celočíselné proměnné. Můžeme si přitom představit, že jednotlivé (celočíselné) hodnoty této proměnné označují různé *stavy* automatu, na němž požadujeme, aby *nacházeje se v libovolném ze svých možných stavů a přijav na svém vstupu libovolný terminální symbol vstupního jazyka, přešel determinovaně opět do některého ze svých stavů.* K tomu, aby bylo možné rozpoznat okamžitě chybu ve vstupním řetězu, tj. příchod symbolu, který signalizuje, že řetěz nebyl sestaven v souhlase

s gramatikou vstupního jazyka, zavádíme ještě další vnitřní stav automatu, který nazveme třeba *alarm*. Přejde-li automat do tohoto stavu, zůstane v něm a signalizuje chybu ve vstupním řetězu.

Funkce popisující chování uvažovaného automatu je funkcí dvou proměnných, a to stavu (celočíselná proměnná) a vstupu (proměnná, jejímiž hodnotami jsou terminální symboly vstupního jazyka). Hodnota funkce pro libovolnou kombinaci stavu a vstupu je opět (následující) stav. Pro přehlednost zakreslujeme závislost charakterizovanou touto funkcí do tabulky (matice), tzv. *přechodové tabulky* (*matice*). Jednotlivé řádky odpovídají jednotlivým stavům a jednotlivé sloupce jednotlivým symbolům vstupního jazyka. Ukážeme si funkci popsaného analyzátoru na příkladě, a to opět z jazyka ALGOL 60.

Uvažme tu část gramatiky jazyka ALGOL 60, která popisuje vytváření čísel. Přitom však budeme předpokládat, že se při analyzování čísel setkáme s řetězy, které budou začínat nějakým počtem (třeba nulovým) zvláštních znaků, tzv. *terminátorů*, za nimiž následuje číslo ve smyslu ALGOLu 60 a za ním opět jeden terminátor. Příslušná pravidla tedy budou (viz [3]):

⟨posloupnost terminátorů⟩ ::= ⟨prázdný⟩ | ⟨terminátor⟩ | ⟨posloupnost terminátorů⟩ ⟨terminátor⟩

⟨celé číslo bez znaménka⟩ ::= ⟨číslice⟩ | ⟨celé číslo bez znaménka⟩ ⟨číslice⟩

⟨celé číslo⟩ ::= ⟨celé číslo bez znaménka⟩ | + ⟨celé číslo bez znaménka⟩ | − ⟨celé číslo bez znaménka⟩

⟨desetinná část⟩ ::= .⟨celé číslo bez znaménka⟩

⟨exponentová část⟩ ::= $_{10}$⟨celé číslo⟩

⟨desetinné číslo⟩ ::= ⟨celé číslo bez znaménka⟩ | ⟨desetinná část⟩ | ⟨celé číslo bez znaménka⟩ ⟨desetinná část⟩

⟨číslo bez znaménka⟩ ::= ⟨desetinné číslo⟩ | ⟨exponentová část⟩ | ⟨desetinné číslo⟩ ⟨exponentová část⟩

⟨číslo⟩ ::= ⟨číslo bez znaménka⟩ | + ⟨číslo bez znaménka⟩ | − ⟨číslo bez znaménka⟩

⟨text⟩ ::= ⟨posloupnost terminátorů⟩ ⟨číslo⟩ ⟨terminátor⟩

Příklady (podrobnější vysvětlení viz [1], str. 160).

Celé číslo: 324, −01005, +6117

Desetinné číslo: 004024, .99, 882.31, 21.0

Číslo bez znaménka: 2.17, $_{10}$891, $_{10}$−0002, $_{10}$+3, 21.987 $_{10}$−18

Číslo: $+21.987_{10} -18$, $00.0021987_{10} -14$, $.21987_{10} -16$, $+_{10} +36$, -0

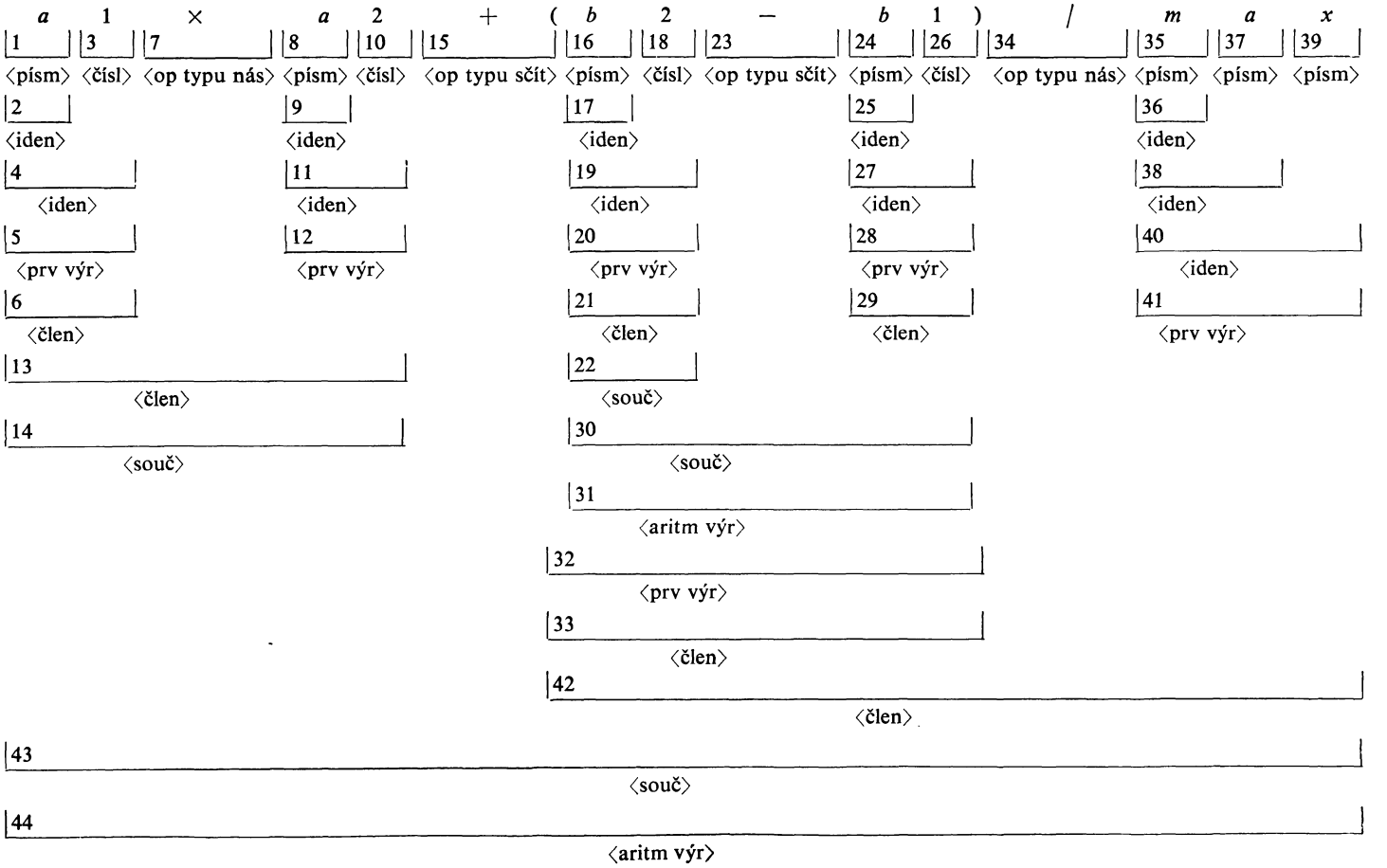
První tři čísla v posledním řádku vyjadřují touž hodnotu. Poslední příklad ve druhém řádku a druhý a čtvrtý příklad ve třetím řádku jsou sice svou hodnotou celá čísla, ale nepočítají se mezi algolovská celá čísla. Zde hraje roli hlavně různý způsob zápisu celých a necelých čísel v paměti samočinných počítačů. Např. $+2_{10} -3$ značí $2 \times \times 10^{-3}$.

Sestavíme si nyní přechodovou tabulku pro kontrolu řetězů z hlediska jejich vlastnosti „být či nebýt textem" ve smyslu výše uvedené gramatiky. Přitom nemusíme mít tolik sloupců, kolik je přitom vůbec možných terminálních symbolů. Některé symboly můžeme spojit ve třídu a mít jeden sloupec pro celou tuto třídu. Dále si u označení každého stavu (celým kladným číslem) napíšeme slovy stručně jeho intuitivní význam. Vystačíme (kromě stavu alarm a speciálních označení pro správný konec analýzy) s osmi stavy.

Stav	Třídy terminálních symbolů				
	znaménko + −	číslice	tečka .	exponentová desítka $_{10}$	terminátor
1: Dosud pouze terminátory	2	3	4	6	1
2: Po prvním znaménku	alarm	3	4	6	alarm
3: Mezi číslicemi před .	alarm	3	4	6	celé číslo
4: Následuje za .	alarm	5	alarm	alarm	alarm
5: Mezi číslicemi po .	alarm	5	alarm	6	desetinné číslo
6: Následuje za $_{10}$	7	8	alarm	alarm	alarm
7: Po znaménku u exponentu	alarm	8	alarm	alarm	alarm
8: Mezi číslicemi v exponentu	alarm	8	alarm	alarm	číslo s exponentovou částí

Sledujme nyní chování automatu, tj. příslušnou posloupnost stavů, na několika příkladech. Jako terminátoru použijeme znaku &. Předpokládáme, že počáteční stav je vždy 1.

Řetěz na vstupu	Posloupnost stavů	Poznámka
&&324&	111333 celé číslo	
−0&	123 celé číslo	
&1.92&	113455 desetinné číslo	
&.01&	11455 desetinné číslo	
$_{10}$−2&	1678 číslo s exp. částí	
+.5$_{10}$+4&	1245678 číslo s exp. částí	
&02.&	11334 alarm	číslo nesmí končit řádovou tečkou
2$_{10}$2.5&	1368 alarm	exponent musí být celé číslo
++13&	12 alarm	smí předcházet nejvýše jedno znaménko


a 1 × a 2 + (b 2 − b 1) / m a x
1 3 7 8 10 15 16 18 23 24 26 34 35 37 39
⟨písm⟩ ⟨čísl⟩ ⟨op typu nás⟩ ⟨písm⟩ ⟨čísl⟩ ⟨op typu sčít⟩ ⟨písm⟩ ⟨čísl⟩ ⟨op typu sčít⟩ ⟨písm⟩ ⟨čísl⟩ ⟨op typu nás⟩ ⟨písm⟩ ⟨písm⟩ ⟨písm⟩
2 9 17 25 36
⟨iden⟩ ⟨iden⟩ ⟨iden⟩ ⟨iden⟩ ⟨iden⟩
4 11 19 27 38
⟨iden⟩ ⟨iden⟩ ⟨iden⟩ ⟨iden⟩ ⟨iden⟩
5 12 20 28 40
⟨prv výr⟩ ⟨prv výr⟩ ⟨prv výr⟩ ⟨prv výr⟩ ⟨iden⟩
6 21 29 41
⟨člen⟩ ⟨člen⟩ ⟨člen⟩ ⟨prv výr⟩
13 22
⟨člen⟩ ⟨souč⟩
14 30
⟨souč⟩ ⟨souč⟩
31
⟨aritm výr⟩
32
⟨prv výr⟩
33
⟨člen⟩
42
⟨člen⟩
43
⟨souč⟩
44
⟨aritm výr⟩

Dá se dokázat zcela korektně ekvivalence mezi popisem čísla pomocí uvedené přechodové tabulky a jeho popisem pomocí přepisovacích pravidel bezkontextové gramatiky ([9]). Postup užitý v důkazu se však nehodí jako vodítko při konstrukci přechodové tabulky z pravidel bezkontextové gramatiky v obecném případě. Tvůrce kompilátoru se v takovém případě řídí spíše svým pochopením a proniknutím do struktury textu. Jen pro zajímavost uveďme, že v kompilátoru pro samočinný počítač GIER se vystačí pro správný algolovský text pouze se 32 stavy celkem při způsobu analýzy navrženém E. W. DIJKSTROU. Pro zjištění chyb v předloženém programu se potřebují ještě další čtyři stavy.

ZÁVĚR

Uvedli jsme si základní myšlenky některých způsobů analýzy textů psaných ve formálních bezkontextových jazycích a ukázali si jejich použití na příkladech jazyků zcela abstraktních i na příkladě konkrétního programovacího jazyka. Musela by být uvedena ještě celá řada detailů a obratů, abychom si mohli popsat realizaci příslušných částí kompilačního procesu na samočinných počítačích. Jde tu např. o výhodný zápis gramatiky v paměti počítače, využití rekurzivity procesů analýzy apod. Čtenáře, který by se chtěl dozvědět více o těchto věcech, odkazuji na některé z posledních sborníků prací „Stroje na zpracování informací", vydávaných ČSAV, a také na cizojazyčnou literaturu, např. sborníky o automatickém programování „Annual Review in Automatic Programming", sv. 1, 2, 3, 4; Pergamon Press.

Vystříhal jsem se v celém článku záměrně zmínky o využití tzv. *sklípkové paměti* přesto, že to je jeden z nejdůležitějších principů používaných při kompilaci, a přesto, že byl implicitně využit i ve zde popisovaných procesech. Domnívám se však, že si zasluhuje samostatné projednání v jiném článku.

Literatura

[1] KOPŘIVA, JIŘÍ, *Automatizace programování.* PMFA *9* (1964), č. 3.

[2] CHOMSKY, NOAM, *On Certain Formal Properties of Grammars.* Inf. and Control, *2* (1959), No 2 (ruský překlad Кибернетический сборник *5*, ИИЛ, Москва 1962)

[3] BACKUS J. W., *Programování v jazyku „Algol 60".* SNTL, 1963.

[4] FLOYD, W. ROBERT, *The Syntax of Programming Languages — A Survey.* IEEE Transactions on Electronic Computers, 1964.

[5] GRIFFITHS, T. V. and PETRICK, S. R., *On the Relative Efficiencies of Context — Free Grammars Recognizers.* Communications of the ACM, *8* (1965), No 5.

[6] IRONS, EDGAR T., *The Structure and Use of the Syntax Directed Compiler.* Annual Review in Automatic Programming, Vol. *3*, Pergamon Press, London, New York 1963.

[7] HOŘEJŠ, JIŘÍ, *Principy samočinných číslicových počítačů.* PMFA *7* (1962), č. 1.

[8] NAUR, PETER, *The Desing of the GIER ALGOL Compiler.* BIT *3* (1963), No 2, 3.

[9] NAUR, PETER, *State Analysis of Linear Texts.* Preliminary draft, June 1965, Regnecentralen Copenhagen.