Acta Universitatis Carolinae. Mathematica et Physica

Ladislav Koubek

Algoritmus překladače z jazyka Algol 60 vhodný pro malý počítač

Acta Universitatis Carolinae. Mathematica et Physica, Vol. 10 (1969), No. 1-2, 47--56

Persistent URL: http://dml.cz/dmlcz/142232

1969 ACTA UNIVERSITATIS CAROLINAE MATHEMATICA ET PHYSICA VOL.10

ALGORITMUS PŘEKLADAČE Z JAZYKA ALGOL 60 VHODNÝ PRO MALÝ POČÍTAČ

L. KOUBEK
Centrum numerické matematiky, UK, Praha

АЛГОРИТМ ТРАНСЛЯТОРА С ЯЗЫКА АЛГОЛ 60, ПРЕДНАЗНАЧЕННЫЙ ДЛЯ МАЛОЙ ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ. В работе приводятся основные принципы, на которых основывается алгоритм транслятора с языка АЛГОЛ 60 на язык вычислительной машины с малой емкостью памяти (около 8.000 ячеек), которая не снабжена никакими внешними запоминающими устройствами.

Даются определения единицы программы, символа конца и уровня, из которых в дальнейших работах выводятся все следующие понятия.

Кроме того, в работе указаны ограничения, налагаемые на язык, вытекающие из алгоритма, предлагаемого автором, а также приводится некоторый опыт с транслятором ФЕН-АЛГОЛ для вычислительных машин ОДРА 1003 и 1013, составленным автором по данному алгоритму.

Ограничений, налагаемых на язык, насчитывается всего 7, однако блочная структура программы вполне сохраняется и возможность рекурсивного обращения к процедурам остается в полном объеме. Не возникает необходимость в спецификации формальных параметров, не входящих в список значений.

V serii prací /2/ - /8/ je podán nástin algoritmu překladače z jazyka ALGOL 60 do kódu počítače. Jde o algoritmus tzv. kompilačního překladače. To znamená, že text napsaný v programovacím jazyku, se překladačem zpracuje v posloupnost strojových instrukcí - program, který je vyděrován na děrné pásce. Při vlastním výpočtu pak program zavádíme obvyklým způsobem do počítače již bez zřetele k tomu, že původní text byl psán v programovacím jazyce ALGOL 60.

Algoritmus je tvořen tak, aby mohl pracovat na jediný průchod, tj. text v programovacím jazyce se načítá po jednotlivých symbolech a ihned zpracovává v program. Symbolem ovšem rozumíme v dalším nikoli základní symboly /1/, nýbrž identifikátory, čísla, řetězy, logické hodnoty a omezovače. Každý symbol je ihned zpracován, tj. zjistíme jeho význam a eventuelně adresu, která je mu přiřazena, provedeme jisté akce a načítáme další symbol.

V prvé fázi, tj. při načítání a zjišťování významu symbolu využíváme přirozeně globální vlastnosti symbolů a musíme proto hledat v seznamech proměnných, polí, procedur, operátorů atd. V další práci algoritmu však už nikde globální vlastnosti nepoužíváme, ale pracujeme jen s lokálními a semilokálními vlastnostmi symbolů /13/.

Prvá fáze načítání symbolů a vyhledávání v seznamech závisí velmi značně na konkrétním použitém počítači a proto ji nebudeme popisovat. Omezíme se jen na to, že budeme předpokládat, že u každého symbolu dovedeme zjistit jeho druh, typ a eventuelně adresu, která je mu přiřazena.

Text v jazyce ALGOL 60 budeme v dalším chápat jako konečnou posloupnost symbolů, které postupně (zleva) načítáme a zpracujeme. V okamžiku, kdy je určitý symbol načítán, je další text zcela nepřístupný a rovněž text, který byl načten dříve je již nepřístupný. V paměti počítače jsou však uloženy některé údaje, které jsme zjistili při zpracování předešlého textu, přesněji řečeno posloupnosti symbolů, které jej vytvářely. Snahou ovšem je, aby těchto údajů, které ukládáme do paměti počítače, bylo co nejméně.

Jak známo /1/, jsou základními syntaktickými jednotkami jazyka ALGOL 60 výraz (a to aritmetický, boolovský a cílový), příkaz a popis.

Popisy zatím ponecháme stranou, neboť v algolském textu je nalézáme jen na počátku bloku a (s vyjímkou popisu procedury) slouží jen k vytváření seznamů a tabulek.

Vzhledem ke zvolenému způsobu práce s textem, při kterém nemůžeme provádět syntaktickou analýzu na základě definic, musíme stanovit symboly, kterými výrazy respektive příkazy mohou končit. Úmyslně mluvíme pouze o výrazech, neboť jak uvidíme v dalším, není nutno striktně rozlišovat mezi výrazy aritmetickými, boolovský-

mi a cílovými. Přitom nám jde o syntaktickou jednotku, tj. o výraz, který je buď pravou stranou dosazovacího příkazu, indexovým výrazem, skutečným parametrem anebo prvotním výrazem. Probereme-li všechny definice odstavců 3 a 4 Zprávy o algoritmickém jazyce ALGOL 60 /1/, zjistíme, že jako syntaktická jednotka v našem smyslu může výraz končit jedním ze symbolů

; | end |] | , | then | else | step | until | while | do (1)

) (1a)

Symboly (1) končí výraz vždy, symbolem (1a) je ukončen právě tehdy, jsou-li uzavřeny všechny závorky v něm obsažené.

Podobně příkaz jako syntaktická jednotka končí jedním ze symbolů

; | end | else

tehdy a jen tehdy, jsou-li uzavřeny všechny příkazové závorky, které se vyskytují v posloupnosti symbolů tvořících příkaz.

Translátor, jehož algoritmus popisujeme v /2/ - /8/, transformuje syntakticky správnou posloupnost symbolů textu v programovacím jazyce na posloupnost strojových instrukcí, které říkáme strojový program. Tuto transformaci provádíme v podstatě po úsecích odpovídajících syntaktickým jednotkám textu. Abychom postup mohli popsat podrobněji, zavedeme následující pojmy, v nichž zpřesníme to, co jsme dosud popisovali heuristicky.

Definice 1: Každý ze symbolů (1) nazveme ukončující symbol.

Komentáře, které jsou do algolského textu zařazovány jen jako vysvětlivky a na vlastní text nemají vliv, translátor nezpracovává a pro naše účely je můžeme pokládat za prázdné symboly, které v dalších úvahách vůbec neuvažujeme.

Popisy, jak uvidíme později v práci /8/, zpracováváme zvláštním algoritmem, neboť jejich úkolem je popsat jisté nelokální vlastnosti identifikátorů, jejichž znalost je při práci algoritmu již nutná a proto je v této práci budeme rovněž pokládat za speciální druh komentářů a každý popis (s vyjímkou popisu procedury), tj. posloupnost symbolů počínající popisovačem a ukončenou středníkem (včetně obou těchto symbolů) nepokládáme za symboly textu. Mluvíme-li v dalším např. o identifikátoru pole, jde vždy o identifikátor zapsaný v textu výrazu nebo příkazu.

Definice 2: Druhý symbol textu (tj. symbol následující za

begin, kterým začíná program), každý ze symbolů if, for, první symbol seznamu indexů, první symbol seznamu skutečných parametrů procedury a konečně každý symbol, který následuje za libovolným ukončujícím symbolem a symboly if, for nazveme počáteční symbol.

Poznámka: V této definici ovšem mlčky předpokládáme, že je dána posloupnost symbolů textu a o tom, zda je symbol počátečním symbolem nebo nikoliv, rozhoduje jeho poloha v této posloupnosti.

Budiž φ libovolný symbol textu a označme b_1 počet symbolů begin, b_2 počet symbolů end, z_1 počet symbolů (a [, z_2 počet symbolů) a], f_1 počet symbolů if a for, f_2 počet symbolů then a do, které jsou před symbolem φ v posloupnosti textu. Rozdíly $B = b_1 - b_2$ a $Z = z_1 - z_2$ udávají zřejmě počet otevřených příkazových závorek resp. obyčejných závorek (mezi které počítáme i závorky hranaté). Význam rozdílu $F = f_1 - f_2$ objasníme až v pracech /4/ a /6/.

Definice 3: Budiž φ libovolný symbol. Úrovní $u(\varphi)$ tohoto symbolu nazveme uspořádanou dvojici

$$u(\varphi) = (B + Z, F). \qquad (2)$$

Z této definice plyne, že úroveň symbolu podstatně závisí na jeho poloze v textu. Přitom úroveň symbolu počítáme vždy v okamžiku jeho načítání, tj. např. u symbolů begin a (před zvětšením b_1 resp. z_1.

Definice 4: Budiž φ_1 počáteční a φ_2 ukončující symbol, přičemž φ_2 je v posloupnosti symbolů textu za φ_1. Řekneme, že symboly φ_1 a φ_2 jsou si přiřazeny když:

1) $u(\varphi_1) = u(\varphi_2)$,
2) je-li φ_1 různé od if, je φ_2 libovolný ukončující symbol,
3) je-li φ_1 symbolem if, je φ_2 různé od else,
4) neexistuje mezi nimi ukončující symbol φ_3 takový, že $u(\varphi_1) = u(\varphi_3)$.

Podle této definice je každému počátečnímu symbolu přiřazen jediný ukončující symbol, opak však neplatí. Jednomu ukončujícímu symbolu může být přiřazeno několik symbolů počátečních.

Definice 5: Jednotka programu je posloupnost symbolů textu začínající počátečním symbolem a ukončená přiřazeným symbolem ukončujícím. Úrovní jednotky rozumíme úroveň jejího počátečního symbolu.

Definice 6: Řekneme, že vzhledem k symbolu φ je jednotka programu otevřená, je-li její počáteční symbol před φ a ukončující symbol za φ. Ostatní jednotky jsou vzhledem k φ uzavřené.

V obdobném smyslu mluvíme při popisu algoritmu o tom, že jednotku otevíráme resp. uzavíráme při nalezení symbolu φ nebo o tom, že dokončujeme její program.

Definice 7: Symbol) je ukončujícím symbolem právě tehdy, když vzhledem k tomuto symbolu existuje alespoň jedna otevřená jednotka programu s počátečním symbolem if a když u()) = u(if).

V dalším vyjdeme z těchto definic a určíme pravidla, podle kterých přiřazujeme jednotkám programu úseky strojových instrukcí. Zásadně při tom používáme jen informace o jednotkách otevřených vzhledem k symbolu, který právě zpracováváme. Informace o uzavřených jednotkách buď již nebo ještě nemáme. Podotkněme ihned, že informace o otevřených jednotkách s nimiž můžeme pracovat, musí být možno odvodit jen z textu, který již byl načten a nesmí se v nich vyskytovat žádný údaj, který by bylo možno odvodit teprve z textu, který následuje za zpracovávaným symbolem.

Z těchto požadavků plyne, že základní definice jsme museli vyslovit poměrně složitě ačkoli vesměs vyjadřují velmi jednoduché a názorné vztahy posloupnosti symbolů. Tak např. program, jehož definice je uvedena v dodatku ke "Zprávě" /1/, je podle našich definic jednotkou programu úrovně (1, 0).

V dalším uvidíme, že většinu vlastností symbolů, které v našich definicích vyžadujeme, není třeba zvlášť zjišťovat, neboť algoritmus je určí zcela automaticky.

Při popisu algoritmu budeme potřebovat kromě seznamů vytvořených při zpracovávání popisů ještě operační tabulku, v níž jsou uvedeny identifikátory všech operátorů (eventuelně identifikátory elementárních funkcí) a u každého z nich je několik údajů, z nichž nejdůležitější je tabulkový stupeň operátoru a posloupnost strojových instrukcí, která je mu přiřazena.

Posledním pojmem, který má pro popisovaný algoritmus základní význam, je hlavní přepínač. Je to jistý přepínač, kterým řídíme další práci algoritmu (po sestavení strojového programu jednotky programu). Hodnotu indexu tohoto přepínače určujeme při načtení

každého ukončujícího symbolu.

Změnou operační tabulky velmi snadno změníme množinu operátorů, které se mohou v textu programovacího jazyka vyskytovat, nebo posloupnosti přiřazených strojových instrukcí a tím způsob jejich provedení na počítači. V našem algoritmu tuto možnost nevyužíváme, neboť v ALGOLu 60 je množina operátorů pevně předepsána a čísla obou typů zobrazujeme v pohyblivé řádové čárce, takže operační tabulku není nutno měnit.

Chceme-li používat (a u mnohých počítačů to je výhodné) zobrazení celých čísel v pevné čárce a čísel reálných v pohyblivé, můžeme podle typu prvého identifikátoru textu jednotky velmi jednoduše zařazovat transformační funkce.

Ještě lépe se tento způsob může uplatnit při překladači z jazyka FORTRAN, kde již nejsou přípustné libovolné kombinace proměnných různých typů ve výrazech, nebo v jazyce ALGOL 68, kde je situace obdobná.

Základní pravidla, podle kterých pracuje algoritmus, vyslovíme v práci /2/ a v dalších pracech je budeme postupně doplňovat tak, abychom dostali algoritmus pro překlad algolského textu. Pravidla vyslovíme tak, aby byla zřejmá možnost, jak změnou některých z nich lze získat algoritmus pro překlad z jiného programovacího jazyka.

Popsaný algoritmus není ovšem použitelný pro text psaný v úplném referenčním ALGOLu 60, nýbrž jen pro text, v němž kromě grafických úprav symbolů podle možností počítače, platí tato omezení:

1) celé číslo bez znaménka nesmí být návěštím
2) prvkem seznamu přepínače může být jen návěští
3) popis identifikátoru (s vyjímkou identifikátoru návěští, který se nepopisuje) musí být uveden, nebo alespoň započat dříve, než je tento identifikátor použit v některém příkazu. Přesněji řečeno, každý identifikátor, který nebyl popsán, je lokalizován v nejmenším bloku, v němž se poprvé vyskytl a je interpretován buď jako identifikátor návěští nebo veličiny typu integer. Jak je možno vyhnout se potížím plynoucím z tohoto omezení je podrobně uvedeno např. v /10/, str. 164, bod b 3.
4) Je-li procedura vyvolávána rekursivně ve své operační části z příkazu cyklu, musí být parametr tohoto cyklu lokalizován

v operační části procedury nebo to musí být formální parametr a seznam cyklů (tohoto cyklu) smí obsahovat jen jediný prvek.

5) Popisovač own je nepřípustný.

6) je-li cílovým výrazem zápis přepínače jehož hodnota není definována, není příkaz skoku ekvivalentní prázdnému příkazu. Jeho provedení vede vždy k těžko kontrolovatelné chybě.

7) Jestliže žádný z boolovských výrazů v podmínce nemá hodnotu true, je účinek neúplného podmíněného příkazu ekvivalentní účinkům, které vzniknou při zjišťování hodnot těchto boolovských výrazů.

Nejsou-li v těchto boolovských výrazech funkční procedury, jejichž operační části mají vedlejší účinky nebo obsahují dosazovací příkazy do některých formálních parametrů, je účinek podmíněného příkazu ekvivalentní účinku prázdného příkazu. Jestliže však ve výrazech takové procedury jsou, budou při výpočtu hodnot boolovských výrazů tyto účinky realizovány, takže účinek celého příkazu nebude ekvivalentní účinkům prázdného příkazu. Obdobná tvrzení ovšem platí i pro výpočet hodnoty podmíněného výrazu.

Jiná omezení z algoritmu neplynou, zejména je zachována blokova struktura programu, možnost rekursivního vyvolávání procedur, a formální parametry vyvolávané jménem není nutno specifikovat.

Omezení 1 je zavedeno proto, že (jak ukázal Petr Naur) nemusí být ve všech případech zřejmo, zda celé číslo bude použito jako konstanta nebo návěští.

Omezeními 2 a 6 docílíme toho, že přepínač lze interpretovat jako speciální lineární pole a není nutné vytvářet zvláštní část algoritmu pro práci s přepínači.

Omezení 3 souvisí s tím, že seznamy vytváříme vždy na začátku bloku po nalezení popisovače a při práci algoritmu nad textem musíme znát vlastnosti každého identifikátoru, což bez předcházejícího popisu není možné.

Omezení 4 je zcela podstatné a nelze ho v našem algoritmu nijak odstranit. Na druhé straně ovšem autor v literatuře dosud nenašel příklad rekursivního vyvolávání procedury z příkazu cyklu v její operační části.

Omezení 5 je nepodstatné a lze ho odstranit velmi snadno a to jak při statickém, tak dynamickém pojetí. Bylo zavedeno hlavně proto, že podle popisovaného algoritmu byl realizován translátor PHEN-ALGOL na počítači ODRA 1003, který nemá vnější paměti a podle autorova názoru mají vlastní veličiny smysl jen při práci s těmito pamětmi.

Omezení 7 je rovněž podstatné a nelze ho v našem algoritmu obejít. Podle autorova názoru by v tomto smyslu měl být upraven text Zprávy /1/, neboť důsledné dodržení jejího textu je téměř nemožné. Pro každou funkční proceduru bychom museli kompilovat dva podprogramy, z nichž jeden by potlačil všechny vedlejší účinky a používal by se jen při vyhodnocování boolovských výrazů v podmínkách.

Podle popisovaného algoritmu byl vytvořen překladač PHEN-ALGOL pro počítače ODRA 1003 a 1013 /9/. Překladač byl ovšem doplněn o vstupní a výstupní procedury, o možnost zařazování procedur ve strojovém kódu a dále o některé standardní procedury, vhodné pro práci na problémech pracovišť Výzkumného ústavu zvukové, obrazové a reprodukční techniky v Praze, pro jehož matematické oddělení je realizovaný translátor především určen. Laskavostí ředitele VÚZORTu dr. M. Jahody, DrSc bylo konkrétní sestavení překladače zařazeno do plánu prací VÚZORTu a tím v podstatě umožněno, neboť velké množství strojových hodin, které si práce na překladači a jeho ověření vyžádaly, nebylo možno jinak získat.

Překladač PHEN-ALGOL má ovšem další omezení, plynoucí z vlastností počítače, jako je např. celková délka programu, rozsah čísel, přípustný maximální počet identifikátorů, atd.

Počítač ODRA byl zvolen z několika důvodů. Především je to počítač s relativně malou operativní pamětí (cca 8.000 slov) a žádné vnější paměti nemá. Obě tyto vlastnosti byly pro autora výhodné, neboť si ověřil, že celý program překladače, včetně všech seznamů, pracovních buněk a polí, se vejde do operativní paměti počítače.

Počítač ODRA 1003 je vybaven dodavatelem, polskou firmou ELWRO, překladačem MOST 1, který je prakticky shodný s u nás velmi rozšířeným a užívaným autokódem firmy ELLIOT (tento autokód byl mimo jiné přenesen, a to v několika verzích, na počítač

MINSK 22). Autorovou snahou bylo prokázat, že jím navržený algoritmus generuje programy, které provádějí výpočty rychlostí, srovnatelnou s rychlostí výpočtů podle programů generovaných autokódem MOST. Podrobný popis celého systému PHEN-ALGOL je uveden v /9/.

Překladač byl v době svého dokončení a uvedení do provozu vyzkoušen na více než 600 příkladech a to jak z technické praxe, tak uměle volených, aby byly vyzkoušeny nejrůznější možnosti programování v ALGOLu. Po zavedení do praxe se s ním pracuje ve dvou nezávislých střediscích a dosud nebyly nalezeny žádné odchylky proti popisu.

Literatura

/1/ Backus J. W. - Programování v jazyku "ALGOL 60", SNTL - Praha 1963

/2/ Koubek L. - Algoritmus kompilace nepodmíněných výrazů, dosazovacích příkazů a příkazů skoku v překladači z jazyka ALGOL 60, AUC, Math.-Phys., Vol. 10, 57-69 (1969)

/3/ Koubek,L. - Zobrazení veličin a specifikace parametrů procedur a jedné realizaci překladače z jazyka ALGOL 60, AUC, Math.-Phys., Vol. 10, 71-76 (1969)

/4/ Koubek L. - Kompilování podmíněných výrazů a příkazů v translátoru z jazyka ALGOL 60, AUC, Math.-Phys., Vol. 10, 77-85 (1969)

/5/ Koubek L. - Překlad proměnných s indexy a přepínačů v překladači z jazyka ALGOL 60, AUC, Math.-Phys., Vol. 10, 87-90, (1969)

/6/ Koubek L. - Programování příkazů cyklu v kompilátoru z jazyka ALGOL 60, AUC, Math.-Phys., Vol. 10, 91-96 (1969)

/7/ Koubek L. - Kompilace popisů a příkazů procedur v kompilátoru z jazyka ALGOL 60, AUC, Math.-Phys., Vol. 10, 97-102 (1969)

/8/ Koubek L. - Vytváření seznamů v translátoru z jazyka ALGOL 60, AUC, Math.-Phys., Vol. 10, 103-104 (1969)

/9/ Koubek L. - Programující program PHEN-ALGOL pro počítače ODRA 1003 a 1013, Závěrečná zpráva číslo 10/69 VÚZORT - Praha

/10/ Raichl J. - Programování v ALGOLu, Academia 1967 Praha

/11/ Randell B. a Russel L. J. - ALGOL 60 Implementation, Ac. Press London 1964 (rus. překlad - Moskva 1967)

/12/ Grau A. A. - The Structure of ALGOL Compiler, ORNL, Oak Ridge 1961

/13/ Kindler E. - Epos - ALGOL compiler, Sborník stroje na zpracování informací - 1965

/14/ Backus J. W. - Report on the Algorithmic Language ALGOL 60, CACM 3 (1960), 299-314

/15/ Revised Report on the Algorithmic Language ALGOL 60, CACM 6 (1963)