Acta Universitatis Carolinae. Mathematica et Physica

Ladislav Koubek

Kompilování podmíněných výrazů a příkazů v translátoru z jazyka Algol 60

Acta Universitatis Carolinae. Mathematica et Physica, Vol. 10 (1969), No. 1-2, 77--85

Persistent URL: http://dml.cz/dmlcz/142235

1969 ACTA UNIVERSITATIS CAROLINAE MATHEMATICA ET PHYSICA VOL.10

KOMPILOVÁNÍ PODMÍNĚNÝCH VÝRAZŮ A PŘÍKAZŮ V TRANSLÁTORU Z JAZYKA ALGOL 60

L. KOUBEK
Centrum numerické matematiky UK, Praha

ТРАНСЛЯЦИЯ УСЛОВНЫХ ВЫРАЖЕНИЙ И ОПЕРАТОРОВ С ЯЗЫКА АЛГОЛ 60. В работе указана полная система правил, по которым проводится компиляция в трансляторе с языка АЛГОЛ 60 всех допустимых типов условных операторов и выражений.

Правила основываются на понятиях уровня и микропрограммы, введенных нами в других работах.

Кроме того здесь показано, что в Сообщении об алгоритмическом языке АЛГОЛ 60, определение оператора "если" излишне ограничено.

Dříve než přistoupíme k popisu algoritmu kompilace podmíněných výrazů a příkazů, tj. dříve než formulujeme pravidla, podle kterých při kompilaci postupujeme, všimneme si některých semilokálních vlastností operátoru postupu if, které sice všechny bezprostředně plynou z odstavců 3.3., 3.4. a 4.3. "Zprávy o algoritmickém jazyku ALGOL 60" /1/, ale pro naši práci mají zásadní význam a proto je uvedeme explicitně.

Každý podmíněný výraz i příkaz začíná operátorem postupu if. Je-li to podmíněný výraz, musí být úplný, takže v dalším textu nutně najdeme operátory then a else (v tomto pořadí), které k němu patří.

Začíná-li operátorem if podmíněný příkaz, může být buď úplný, tj. v dalším textu najdeme oba operátory then i else, nebo neúplný,

tj. nalezneme jen odpovídající then ale nikoliv else.

Výraz nebo příkaz za then nesmí být podmíněný. Odtud plyne, že symbolem, který v textu následuje za then, nesmí být operátor if.

Z těchto vlastností podmíněných výrazů ihned plyne, že podmíněný výraz obsahuje všechny tři operátory if, then, else a jako jednotka programu může být ukončen jedním ze symbolů:

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

nemůže však být ukončen symbolem else.

Podmíněný příkaz obsahuje buď všechny tři operátory if, then, a else, nebo jen první dva a může být ukončen jen jedním ze symbolů

; | end (2)

Jedním ukončujícím symbolem (1) může být ovšem ukončeno i několik (podmíněných) výrazů. Např.

n + (if a<b then - 1 else if b<c then 3 else 4) (3)

V tomto příkladu symbol) ukončuje dva podmíněné výrazy.

Podobně u podmíněného příkazu

if a<b then x:= 0 else if a<0 then x:= 1; (4)

Středník ukončuje úplný podmíněný příkaz, neúplný podmíněný příkaz a samozřejmě nepodmíněný výraz 1.

Naopak, nalezneme-li ukončující symbol, nemusí to vůbec znamenat, že podmíněný příkaz končí. Např. v podmíněném příkazu

if a<b then begin x:=1;y:=2 end;

první středník ani end neukončují příkaz.

Všechny tyto poznatky jsou precisovány v práci /2/ v definicích 1 - 7.

Použijeme-li např. výraz (3) jako pravou stranu dosazovacího příkazu a předpokládáme-li, že úroveň počátečního symbolu příkazu je (α,β), dostaneme:

x:=n + (if a < b then - 1 else
(α,β) $(\alpha+1,\beta)(\alpha+1,\beta+1)(\alpha+1,\beta+1)(\alpha+1,\beta)(\alpha+1,\beta)$ (5)
if b < c then 3 else 4) ;
$(\alpha+1,\beta)(\alpha+1,\beta+1)(\alpha+1,\beta+1)(\alpha+1,\beta)(\alpha+1,\beta)(\alpha+1,\beta)(\alpha+1,\beta)(\alpha,\beta)$

kde pod každý symbol jsme zapsali jeho úroveň. Podle zmíněných definic lze tento příkaz rozložit na následující jednotky programu

(píšeme je v pořadí, v němž jsou při práci algoritmu postupně uzavírány a to s uvedením úrovně a ukončujícího symbolu).

```
1:  (α+1,β+1) : a<b then
2:  (α+1,β)   : - 1 else
3:  (α+1,β+1) : b<c then
4:  (α+1,β)   : 3 else
5:  (α+1,β)   : 4 )
6:  (α+1,β)   : if b<c then 3 else 4 )
7:  (α+1,β)   : if a<b then - 1 else if b<c
                then 3 else 4 )
8:  (α+1,β)   : x:= n + (if a<b then - 1 else
                if b<c then 3 else 4 ) ;
```

Dílčí programy jednotek 1 - 5 sestavíme algoritmem, který jsme popsali v /3/. Zbývá stanovit pravidla, jak z těchto dílčích programů složit program výpočtu jednotek 6, 7 a 8. Abychom tato pravidla snáze odvodili, rozebereme nejprve jednodušší případ dosazovacího příkazu, jehož pravou stranou je podmíněný výraz:

```
x:= if a then b else c;
```

kde a je boolovská proměnná.

Je-li (α,β) úroveň této jednotky programu, můžeme ji opět rozložit na menší jednotky, které sepíšeme tak, jak budou uzavírány.

```
1:  (α,β+1)  a then
2:  (α,β)    b else
3:  (α,β)    c ;
4:  (α,β)  if a then b else c ;
5:  (α,β)  x:= if a then b else c ;
```

Uvážíme-li, že logická hodnota true je zobrazena číslem + 1 a false číslem - 1 a označíme-li T strojovou instrukci podmíněného předání řízení při záporné hodnotě, U strojovou instrukci nepodmíněného předání řízení, je vidět, že strojový program jednotky 4 sestavíme z programů jednotek 1, 2 a 3 tak, že za program jednotky 1 zařadíme instrukci T φ, kde φ je adresa první instrukce jednotky 3 a za program jednotky 2 zařadíme instrukci U σ, kde σ je adresa instrukce Hx, nebo obecněji řečeno, adresa první instrukce programu, který následuje za programem jednotky 4. Adresy φ a σ ovšem v okamžiku, kdy uzavíráme dílčí jednotku ještě neznáme, ale můžeme je určit ihned po dokončení programu jednotky 4. Tzn., že při programování podmíněných výrazů a příkazů musíme

postupovat tak, že postupně sestavíme z mikroprogramů dílčí programy "nepodmíněných" jednotek programu, na místa operátorů then zařadíme instrukce T s neurčenými adresami φ a na místa operátorů else instrukce U s neurčenými adresami δ. Tyto adresy dodatečně dosadíme po uzavření celé "podmíněné" jednotky.

Ze syntaktických pravidel ALGOLu 60 plyne, že podmíněné výrazy můžeme po uzávorkování používat jako prvotní výrazy. Musíme proto počítat s tím, že výsledek aritmetického výrazu bude použit ve vyšší programovací jednotce jako proměnná. Náš algoritmus je při tom volen tak, že podmíněné výrazy jsou kompilovány (a jejich programy dokončeny) před sestavováním programů jednotek, jejichž částmi jsou. Proto musíme hodnoty podmíněných výrazů dosazovat nejprve do zvláštního pole pracovních buněk f strojového programu. Toto pole lze ovšem při realizaci na počítači velmi jednoduchými úpravami algoritmu spojit s pracovním polem p. V dalším f_i znamená prostě i-tou buňku tohoto pracovního pole f.

Vlastní algoritmus nejprve heuristicky popíšeme a pak teprve vyslovíme pravidla.

Zvolme pevně index m s počáteční hodnotou 0 a 7 lineárních pracovních polí (kompilátoru). Označme je J, O, M, F, Z, T, E. Při realizaci na počítači lze jejich počet snížit, neboť několik údajů lze umístit do jedné buňky. Například v PHEN-ALGOLu jsou tato pole jen tři.

Nalezneme-li symbol if, zvětšíme m a zapíšeme do těchto polí (rozumí se do m-tých buněk) údaje:

J_m := úroveň symbolu if
O_m := instrukci φ_{r-1} a její stupeň s (φ_{r-1})
M_m := index mikroprogramu, do kterého právě zapisujeme
F_m := nejmenší z indexů mikroprogramů, které ještě nebyly otevřeny
Z_m := relativní adresu poslední obsazené buňky mikroprogramu M
T_m := 0
E_m := 0

Do S dosadíme instrukci B stupně 0 a do A adresu nuly.

Poznámka: Zde i v dalším textu používáme termíny a označení z práce /3/.

V dalším pracujeme stejně jako v případě kompilace nepodmíněných výrazů, tj. nalezneme-li libovolný ukončující symbol, dokončíme programování (mikroprogramy sestavíme do programu), ale (viz /3/) končíme přepsáním mikroprogramu, jehož index je v F_m. Dosazovací instrukce z mikroprogramu M přepisujeme jen tehdy, jsou-li v buňkách s relativními adresami většími než údaj v Z_m.

Další práce se pak větví.

Je-li ukončujícím symbolem else, zařadíme do programu prázdnou instrukci a její vnější adresu α zapíšeme do E_m.

Je-li ukončujícím symbolem then, zařadíme do programu prázdnou instrukci a její vnější adresu β zapíšeme do té pracovní buňky pole T, která je dosud nulová a má nejmenší index.

Je-li ukončujícím symbolem jiný smybol než then nebo else, zjistíme nejprve, je-li jeho úroveň rovna údaji v J_m, (u then toto zjišťování provádíme až po provedení předchozích akcí).

Je-li úroveň $u \neq \langle J_m \rangle$, dosadíme do A adresu nuly, do S instrukci B, je-li potřeba, změníme hodnotu úrovně, a pokračujeme v kompilaci podle pravidel nepodmíněného výrazu.

Je-li $u = \langle J_m \rangle$, dosadíme instrukce do dosud neobsazených buněk programu:

Je-li $\langle E_m \rangle \neq 0$, zapíšeme do buňky programu s vnější adresou $\alpha = \langle E_m \rangle$ instrukci $U\sigma$, kde σ je adresa první dosud neobsazené buňky programu. Do buňky s vnější adresou $\beta = \langle T_m \rangle$ zapíšeme instrukci $T\alpha + 1$.

Je-li $\langle E_m \rangle = 0$, zapíšeme do buňky s vnější adresou β instrukci $T\sigma$.

Pak zmenšíme m a postup opakujeme, pokud buď $u \neq \langle J_m \rangle$ nebo $m = 0$.

Po nalezení m, pro které $u = \langle J_m \rangle$, dosadíme do S operátor uschovaný v O_{m+1}, a je-li ukončujícím symbolem), zařadíme do programu instrukci Hf_{m+1}. Do A zapíšeme adresu pracovní buňky f_{m+1}, provedeme úpravu úrovně a pokračujeme v kompilaci, přičemž budeme zapisovat do mikroprogramu, jehož index je v M_{m+1}.

Popišme jen zcela stručně průběh kompilace dosazovacího příkazu (5).

Do polí J,... zapíšeme při nalezení symbolů:

if:	J_1:=α+1,β	O_1:=+	M_1:=0	F_1:=1	Z_1:=1
	T_1:=0	E_1:=0			
then:	T_1:=φ+2				
else:	E_1:=φ+5				
if:	J_2:=α+1,β	O_2:=B	M_2:=0	F_2:=1	Z_2:=1
	T_2:=0	E_2:=0			
then:	T_2:=φ+8				
else:	E_2:=φ+10				

Mikroprogramy budou (uvádíme název mikroprogramu, symbol, při kterém mikroprogramy přepisujeme do programu a posloupnost strojových instrukcí mikroprogramu):

		then:	**else**:	**then**:	**else**:	):
M	$M_0^{s(+)}$	M_1^0	M_1^0	M_1^0	M_1^0	M_1^0
Hx	Bm	Ba	B0	Bb	B3	B4
		$+f_1$	<b	-1	<c	

Program, který postupně sestavíme, je:

vnější adresa:	instrukce:	instrukce doplněná po nalezení):
φ:	Ba	
φ+1:	<b	
φ+2:	--	Tφ+6
φ+3:	B0	
φ+4:	-1	
φ+5:	--	Uφ+12
φ+6:	Bb	
φ+7:	<c	
φ+8:	--	Tφ+11
φ+9:	B3	
φ+10:	--	Uφ+12
φ+11:	B4	
φ+12:	Hf_1	
φ+13:	Bn	
φ+14:	$+f_1$	
φ+15:	Hx	

Z právě popsaného postupu je vidět, že když index m>0, musíme při nalezení každé uzavírající závorky zjišťovat, je-li ukončujícím symbolem. Stačí ovšem uvažovat jen údaj z J_m pro největší m.

Nyní už můžeme vyslovit pravidla kompilace. Abychom je mohli

formulovat co nejstručněji a bez vyjímek, budeme ještě předpokládat, že $\langle J_0 \rangle = (-1,0)$, $\langle M_0 \rangle = 0$, $\langle F_0 \rangle = 0$, $\langle Z_0 \rangle = 0$, $\langle 0_0 \rangle = B$ a že adresa nuly je f_0. Hodnotu úrovně symbolu φ označíme u (φ) a počítáme ji vždy v tom okamžiku, kdy je symbol načítán.

Nejprve upravíme pravidla B a C 1. Pravidlo C 1' nahradíme pravidlem C 2 a pak vyslovíme další pravidla (původní pravidlo C 2 zrušíme).

Pravidlo B: Do programu zařazujeme mikroprogramy sestupně podle indexu i. Je-li $i \geq 1+\langle F_m \rangle$ a první buňka mikroprogramu M^k_{i-1} je nenulová, zařadíme instrukci Hp_i. Zařazování ukončíme po zapsání $M^k_{F_m}$. Je-li alespoň jedna buňka mikroprogramu M s relativní adresou větší než Z_m nenulová, pak když:
a) $\langle Q \rangle = 0$, $\langle Q1 \rangle = 1,2$ zařadíme program transformační funkce a přejdeme k bodu d.
b) $\langle Q \rangle = 2$, $\langle Q1 \rangle = 1, 2$ zařadíme modifikovaný program transformační funkce a přejdeme k d.
c) $\langle Q \rangle = 1$ nebo $\langle Q1 \rangle = 0$ přejdeme k d.
d) přepíšeme všechny instrukce mikroprogramu M s relativní adresou větší než Z_m do programu, vymažeme příslušné buňky mikroprogramu M a pokračujeme podle pravidla D (t).

Pravidlo C1: Načteme-li symbol (, zvětšíme hodnotu z a první souřadnici úrovně. Pak pokračujeme podle A 0.

Pravidlo C2: Načteme-li symbol), zmenšíme z. Je-li $u()) \neq J_m$, pokračujeme podle A 0. Je-li $u()) = J_m$, pokračujeme podle A 1.3.1.

Pravidlo C3: Načteme-li symbol <u>if</u>, zvětšíme index m a dosadíme:
J_m:= u(if); 0_0:= S; M_m:= index mikroprogramu, do kterého právě zapisujeme;
F_m:= minimální index neotevřených mikroprogramů;
Z_m:= relativní adresa poslední obsazené buňky mikroprogramu M; T_m:= E_m:= 0.
Do pracovní buňky A dáme adresu nuly, do S instrukci B.
Pak zvětšíme druhou souřadnici úrovně a pokraču-

jeme podle A O.

Pravidlo D (else): V programu vynecháme instrukci a její vnější adresu zapíšeme do E_m. Pak pokračujeme podle A O.

Pravidlo D (then): V programu vynecháme instrukci a její vnější adresu zapíšeme do T_j, kde $j \leq m$ je největší index, pro který $T_j = O$. Pak pokračujeme podle pravidla D O. Po jeho dokončení zmenšíme druhou souřadnici úrovně a pokračujeme podle A O.

Pravidlo D ()): Do programu dosadíme instrukci Hf_m a pracujeme podle D O. Po jeho ukončení zmenšíme první souřadnici úrovně a pokračujeme podle A O.

Poznámka: Pravidlo D ()) bude později ještě nutno upravit, protože uzavírající závorka může ukončovat také seznam parametrů procedury.

Pravidlo D (end): Pracujeme podle D O, po jeho ukončení zmenšíme první souřadnici úrovně a pokračujeme podle A O.

Pravidlo D (;): Pracujeme podle D O a po jeho ukončení podle A O.

Pravidlo D O: D O.1. Je-li $u(\varphi) \neq J_m$, dosadíme do A adresu f_i, do S instrukci z O_m a opravíme index mikroprogramu, do něhož budeme zapisovat (údajem z M_m). Pak pokračujeme podle pravidla D (t), které odpovídá nalezenému ukončujícímu symbolu.
D O.2. Je-li $u(\varphi) = J_m$, dosadíme instrukce do dosud neobsazených buněk programu takto: Je-li $\langle E_m \rangle \neq O$, zapíšeme do buňky s adresou $\langle E_m \rangle$ instrukci $U\sigma$ kde σ je adresa instrukce Hf_i. Do T_m dosadíme instrukci $T \langle E_m \rangle + 1$.
Je-li $\langle E_m \rangle = O$ a $\langle T_m \rangle > O$, zapíšeme do T_m instrukci $T \sigma$.
Je-li $\langle T_m \rangle \leq O$, postupujeme podle pravidel, která formulujeme později.
Zmenšíme m a opakujeme postup od D O.1.

Poznámka: Uzávorkováním podmíněných výrazů můžeme do programu zařadit dvě zbytečné instrukce Hf_m a Bf_m, neboť uzavírající závorka se stane ukončujícím symbolem a podle pravidla D ()) dosadíme do programu instrukci Hf_m. Při dokončování programu jednotky programu po nalezení dalšího ukončujícího symbolu, zapíšeme nejprve do M_1^0 instrukci Bf_m, kterou pak přepíšeme do programu. Eliminovat tyto instrukce není obtížné, ale práce algoritmu se značně zpomalí.

Je zajímavé, že náš algoritmus dovoluje kompilovat správně i příkazy, jejichž použití je v jazycích nižší úrovně běžné, podle syntaxe jsou však v ALGOLu 60 vyloučeny. Např výraz

if a then if b then x:=0 ;

(kde a a b jsou boolovské výrazy), je podle syntaxe ALGOLu 60 chybný, naším algoritmeme však bude kompilován správně (tj. tak, jak byl programátorem míněn). Podle autorova názoru jde v tomto případě o zbytečné omezení ALGOLu 60, neboť by stačilo v odstavci 4.5.1. "Zprávy o algoritmickém jazyku ALGOL 60" /1/ změnit definici podmíněného příkazu takto:

⟨Podmíněný příkaz⟩ := ⟨příkaz "když"⟩ | ⟨příkaz "když"⟩ else ⟨příkaz⟩ | ⟨podmínka⟩ ⟨příkaz cyklu⟩ | ⟨podmínka⟩ ⟨příkaz "když"⟩ |⟨návěští⟩ : ⟨podmíněný příkaz⟩

Naším algoritmem je opět možno zpracovávat všechny druhy výrazů všetně výrazů cílových, průběžná kontrola syntaktické správnosti textu je však téměř vyloučena.

Literatura

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

/2/ Koubek L. - Algoritmus překladače z jazyka ALGOL 60, AUC, Math.-Phys., Vol. 10, 47-56 (1969)

/3/ Koubek L. - Algoritmus kompilace nepodmíněných výrazů, dosazovacích příkazů a příkazu skoku v translátoru z jazyka ALGOL 60, AUC, Math.-Phys., Vol. 10, 57-69 (1969)

/4/ Koubek L. - Vytváření seznamů v translátoru z jazyka ALGOL 60, AUC, Math.-Phys., Vol. 10, 103-104 (1969)

/5/ Koubek L. - Programování příkazů cyklu v kompilátoru z jazyka ALGOL, AUC, Math.-Phys., Vol. 10, 91-96 (1969)