IV-01

情報処理学会第27回(昭和58年後期)全国大会

コンパイラ構成法の比較

木村　友則*　　武市　正人**

6三-9

(*東京大学・大型計算機センター，　**電気通信大学・計算機科学科)

1．はじめに

一般にコンパイラの作成を行なうばあいには，一度構成法を決めて作成すると，もう別の方法で作成することは行なわれない．ここでは，幾つかの方法で同じ言語のコンパイラを作成したときに，どのような差があるのかを調べた結果について報告を行なう．実際に実用的な対象言語を設定し，語彙解析部と構文解析部を２通りに構成し比較・検討を行なった．いずれの方式でも"標準的で自然である"と考えられるプログラムを作るように留意した．実験はVAX11/780とM280Hの２種類の計算機上で行なった．

また，C のレジスタとマクロの効果についても，調査を行なった．

2．コンパイラの構成の仕方

比較のために作成するコンパイラは，図１のように，語彙解析部は，自然にtop-down に書き下したものと，LEXで生成したものを用い，構文解析部はtop-down recursive descent に書いたものと，YACCによって生成したものを用いる．構成としては，①top-down+recursive descent+...と②LEX+YACC+...の２通りのコンパイラを作成する．いずれも構文主導型の翻訳方式である．①の方式はPascalで実現し，②の方式はCで実現する．しかし，異なる言語によって作成したばあい，速度の比較は難しいので，あとで①を Cで書換えての実験も行なう．目的とする言語は PLAN〔1〕をシステム記述言語風に少し修正したものを用いる（PLAN2と呼ぶことにする．）．PLAN2は比較的小さい言語であるが自分で自分自身を記述できる程度の能力をもつ．コンパイラは１パス方式とし，目的コードとしてOCODE〔4〕を出力する所までを作成する．実行は BCPLの処理系のOCODEの機械語生成部と実行環境とを用いて行なう．

	書き下しによる	生成系による
語彙解析部	top down	LEX
構文解析部	recursive descent	YACC

図１

3．コンパイラの作成

(１)語彙解析部

構文解析部から見た語彙解析部は，１つの手続き（Pascal版）または，関数（LEX, YACC版）となっている．tokenの種類と，必要に応じて属性などを表わす２つの全域変数を用いる．LEX, YACC版の場合には，tokenの種類は関数の値として返す．

(２)構文解析部

YACCによる構文解析では，構文のなかに動作を Cのプログラムとしてかくことができるのでそれを利用する．Pascalの版では，いきなり構文解析部を作成してしまうかわりに，まず，構文規則どうりに tokenを受理するacceptorを作る（これをSKELETONと呼ぶことにする．）．その後，構文の検査と誤り回復をする部分を付け加える（これをSYNと呼ぶ）．SYNは SKELETONより機械的に作ることができる．その後，SYNに意味解析部とコード生成部を付け加えて目的のコンパイラが出来上がる．

(３)表の構成

PLAN2 は，名前の有効範囲の規則も単純で，また，型もないので表の構造も単純にすることができるが，ここでは，実用の十分な速度が得られる範囲内でできるだけ一般的に表を構成する．PLAN2 の名前表はハッシュ表で実現する．これらの表へのデータの登録，書き込みおよび探索はすべてそれを行なうための手続きあるいは関数を通して行なう．従って表を別の形式に変えることは容易にできる．

これらの表は，Pascal版では配列と添字として扱い，LEX, YACC 版および C版では，配列とレジスタで実現する．また，後で作成するCのrecursive descent版のばあいには，表のポインタなどをレジスタ変数で置き換えたり，表の読み書きの手続きや関数をマクロで置き換える実験を行なう．

(４)予約語の扱い

予約語は初期値設定ルーチンで表に登録しておくこともできる．予約語の処理に表を用いるかは，インプリメントのしかたに応じて決定する．Pascal版では 表を用いるが，LEX版ではオートマトンで予約語を見付けることにする．

(５)目的コード

OCODEは 仮想のスタック計算機を仮定した（中間）コードであるから，コード生成部の負担が軽くなると同時に，自然にコード生成部を記述できるという利点がある．手続きの呼び出しの仕方はおおよそBCPLに従うが，ネストした内側の手続きのなかで外の手続きに局所的な変数を参照するために，静鎖(static link)を使用する．

(６)テストとデバッグ

最後に，PLAN2コンパイラをPLAN2自身で記述してみる．これは，デバッグおよびテストデータとして用いる．

4．作成したコンパイラに関する比較

（1）数量的比較

①大きさの比較：　ソースプログラムの大きさと，それぞれのコンパイラの目的語での大きさは，図2のようになった．ソースプログラムは行数としてはあまり差はない．目的語の大きさは，LEX+YACC版とCによるtop-down recursive descentとを比べると，多少差があるようだが有意な差は見られない．

②コンパイラの速度：　図3は各コンパイラの翻訳の速さをそれぞれVAX11/780と M280Hとでの CPU時間を計測したものである．LEX+YACCのほうがtop-down+recursive descentに比べると約2倍遅いことがわかる．また，PLAN2 自身で記述したコンパイラは，十分使用可能な速度で動いていることがわかる．これはBCPLの機械語生成部の性能によるものと考えられる．

③Cにおけるレジスタ変数とマクロの有効性：　図4は，Cのtop-down+recursive descent版を使って，レジスタ変数＋マクロ，レジスタ変数，マクロ，いずれも使用しないの4つのばあいについて速度を調べた結果である．VAX11/780 では，マクロは効果があるようだが，レジスタ変数は効果が見られない．一方，M280Hでは，レジスタ変数およびマクロの両方で効果が見られる．

（2）非数量的比較

①語彙解析部の作り易さ：　記述の行数にはあまり差は無いが，LEX を使って語彙解析部を作るばあいほとんど規則を並べるだけであり，正規表現に慣れていれば数時間で作成でき誤りが起こりにくい．一方，top-downに記述するばあい1日あるいはそれよりも時間がかかり，誤りをおかしやすい．

②構文解析部の作り易さ：　YACCのばあい 構文をLALR(1)文法で記述しなくてはならないので慣れが必要であり，なかなか難しいが記述量は少なくてすむ．一方，recursive descentのばあい，BNFなどの記述からごく自然に作ることができるが仕事量が少し増える．

③構文誤りの処理：　YACCについての深い知識が必要である．recursive descent のばあい手間をかければできるが，それでも完全なものにするのは難しい．

5．おわりに

実験を始めるときには，generator を使った記述のほうが簡潔になることを期待したが，実際にはそれほど短くはならなかった．これは意味解析・コード生成部が占める割合がかなり多いことにあると思われる．

コンパイラを作成するばあいコンパイラの処理速度が速いことは重要である．しかし，それと同時にコンパイラの信頼性，保守・修正・作成の容易性も大切である．YACCや LEXなどの generatorを使ったばあい，たまたま速度が遅いという結果がでたが，確かに作成の容易性，保守性においては優れていると思われる．今後は属性文法などで表現したばあいどのようになるのかについても調査したい．

使用言語	VAX/UNIX	M280H/VOS3
Pascal　(1277行)	34816	93640
LEX+YACC(1045行)	23552	79069
C　(1159行)	19456	79488
PLAN2　(1274行)		69488

図2　目的語の大きさ（単位＝バイト）

VAX11/780	語彙解析	構文解析*	計
Pascal	11.6	6.6	18.2
LEX+YACC	4.4	3.0	7.4
C	1.7	2.1	3.8

M280H	語彙解析	構文解析*	計
Pascal	0.80	0.85	1.65
LEX+YACC	0.38	0.33	0.70
C/370	0.20	0.15	0.36
PLAN2	0.22	0.31	0.53

図3　コンパイラの速度（単位＝秒）
（*構文解析＋意味解析＋コード生成）

	VAX11/780	M280H
ﾚｼﾞｽﾀ + ﾏｸﾛ	3.8	0.36
ﾚｼﾞｽﾀ	4.7	0.43
ﾏｸﾛ	4.8	0.49
なし	4.8	0.49

図4　レジスタとマクロの効果（単位＝秒）

<参考文献>

〔1〕武市正人，ミニ言語のミニコンパイラ，Bit, vol. 6, nos. 10-12(1974).

〔2〕S. C. Johnson, 'Yacc:Yet Another Compiler-Compiler', Unix programmer's manual 7th edition, Bell telephon laboratories.

〔3〕M. E. Lesk and E. Schmidt, 'Lex - A Lexical Analyzer Generator', Unix programmer's manual 7th edition, Bell telephon laboratories.

〔4〕M. Richards, 'The portability ofthe BCPL compiler', Software-Practice and Experience, vol. 1, no. 2(1971).