C言語プログラムのための複雑性評価方式の検討と実験

4H－1

宮武 圭子　瀧塚 孝志　小花 貞夫　Hannu Aronsson*
国際電信電話株式会社 研究所，Helsinki University of Technology*

1. はじめに

プログラムの開発工数は仕様書の完全性や開発者の能力に依るものが大きい．開発工数を客観的に評価するために，プログラムの複雑性評価方式の研究が進められてきた[1][2][3]．しかし，従来の評価方式はFortranやAlgol等の言語を対象としたものであり，C言語を対象としたものではなかった．また，これらはプログラム構造によっては過大評価を行う等の問題点があり，十分な評価が行えなかった．

そこで，C言語における最適な複雑性評価方式を検討するため，複雑性評価プログラムを実装し，既存のプログラムを対象とした実験を行ったので報告する．

2. 従来の複雑性評価方式

現在よく用いられる複雑性評価方式を以下に示す．

2.1 プログラムサイズ

プログラムの工数評価には，ソースコードの行数や実行文数，オブジェクトコードのサイズ等がよく用いられている．C言語では，コメント文やプログラムの書式によって，ソースコードの行数が変わってくる．オブジェクトコードのサイズは機種やOSによって異なる．実行文の数は，ソースコードの行数やオブジェクトコードのサイズよりも工数を反映する尺度であるが，カンマ演算子やマクロ定義の使用によって実行文の数も一定でない．そのため，書き方に依存しない正規化した実行文数を求める方法が必要である．

2.2 McCabeのCyclomatic Number

プログラムの制御構造に基づき，式(1)により関数毎にCyclomatic Number，$v(G)$を求め，その合計をプログラムの複雑性とする[1]．

$$v(G) = \text{条件分岐の数} + 1 \quad (1)$$

対象言語はFortranである．case文等の多方向分岐を過大評価するという問題点が指摘されている．

2.3 Halsteadの Software Effort

演算子や被演算子等の識別子の総数Nとその種類nからプログラムの開発時間Tを式(2)により関数毎に求め，その合計をプログラムの複雑性とする[2]．

$$T = \frac{N^2 log_2 n}{4S} \quad (2)$$

ここでSはStroud Number(識別個数)で，$5 < S < 20$の定数である．Stroud Numberは1秒あたりに人間が識別子を識別する個数であり，プログラムを読む人や言語に依存する．4章の実験では、文献[2]に従い$S = 18$とした．全ての識別子が$log_2 n$の識別をしているという前提の基に，nとNの間には次の関係が成り立つ．

$$N = n log_2(n/2) \quad (3)$$

対象言語はAlgolの初期のタイプである．

2.4 システム複雑性

内部複雑性ICと外部複雑性ECの加重平均から関数毎に複雑性SCを求め，その自乗平均和をシステム複雑性TSCとする[3]．

$$TSC = \sqrt{\sum (SC)^2} \quad (4)$$

外部複雑性は，呼び出している関数の数をmとして，$EC = 0.5m$である．内部複雑性には実行文の数またはCyclomatic Number，$v(G)$を用いる．実行文の数を用いた場合は式(5)，$v(G)$を用いた場合は式(6)である．

$$SC_1 = 0.5IC + 0.5EC \quad (5)$$

$$SC_2 = 0.1IC + 0.9EC \quad (6)$$

3. 複雑性評価プログラムの実装

前章で述べた尺度をC言語に適用した場合の最適な係数を求めるため，これらの方式を用いた複雑性計算プログラムをSun SparcStation2のUNIX上で実装した．その特徴を以下に示す．

3.1 中間コードRTL(レジスタ転送言語)の採用

前章の評価方式を用いて複雑性を計算するために，正規化された実行文数，条件分岐の数，演算子や被演算子の総数，使用している関数の数を求める必要がある．このときの問題点を以下に述べる．

(1) プログラムの複雑性はソースコードの複雑性を評価すべきであり，マクロ定義は最初の1回だけ展開することが望ましい．しかし，マクロを展開しないで評価すると，構文解析が困難であるとともに，GNUのCコンパイラであるGCC[4]では，マクロの外に分岐するGOTO文等が使用されており，制御構造が実際の構造と異ってしまう場合もある．そこで，マクロは展開することにした．

Study on Complexity Measure for C Programs
Keiko MIYATAKE, Takashi TAKIZUKA, Sadao OBANA, Hannu Aronsson*
KDD R&D Laboratories
2-1-15 Ohara, Kamifukuoka, Saitama 356, Japan
Helsinki University of Technology, Finland*

(2) C言語では "x = y++;" と "x = y; y++;" のように，実行文数が異なっていても同じ動作をするプログラムを書くことができる．そこで，中間コードの個数を実行文数として採用することにした．

GNU の C や PASCAL 等のコンパイラは，構文木を言語に依存しない中間コード，RTL(レジスタ転送言語) に変換し，フロー解析，各種最適化，コード生成のフェーズに分け，処理を行っている．RTL は Lisp 風のデータ構造を持ち，仮想レジスタを使用した仮想の命令を表している．マクロを展開後の評価をすることと，中間コードを得られることから RTL を使用することにした．

3.2 フェーズの選択

GCC は各フェーズ毎に RTL を出力することができる．命令の合成や分岐の簡約化などの最適化により，出力される RTL の個数や制御構造が変わる．本プログラムでは，基本ブロック (Basic Block) の情報を取るため，フロー解析後の RTL を用いることにした．

3.3 RTL からの換算

- McCabe の Cyclomatic Number
  フローグラフの条件分岐の数を数えるため，(各基本ブロックに入って来るエッジの数) − (基本ブロック数) + 2 により求める．
- Halstead の Software Effort
  RTL ではソースコードレベルの演算子と被演算子を完全に抽出することはできないため，文献[5]により，演算子や被演算子の数 N を機械語の数 P から $N = \frac{8}{3}P$ で見積もることにした．
  SparcStation 上では，RTL の数 R の約 1.25 倍の機械語 P が生成されるので，機械語の数は $P = 1.25R$ とした．

3.4 オプション機能

ネストしたプログラム構造内の命令に高い複雑性を与えるため，オプションで指定された場合に比重付きサイズ[6]を RTL の数 R の代りに使用し，(比重付きサイズ) $= R + (if$文のレベル$) + 1$ で計算する機能を持たせた．

4. 実験と考察

4.1 複雑性評価プログラムの適用実験

表1に示す，GNU の bison-1.22 のプログラムを対象として実験を行った．これらを用いて，RTL ダンプを解析した結果のシステム複雑性 1 SC_1，システム複雑性 2 SC_2，Software Effort SE，$v(G)$，関数の総数を表2に示す．

表 1: サンプルプログラム

プログラム	行数	文字数	RTL 数
LR0.c	2195	15530	1113
closure.c	1043	6750	544
conflicts.c	2160	15478	2470
getopt.c	3281	21565	1141
lalr.c	2008	13874	1957

表 2: 実行結果

プログラム	SC_1	SC_2	SE	$v(G)$	関数
LR0.c	41.3	22.1	28.9	91	19
closure.c	26.0	15.7	11.4	50	8
conflicts.c	89.1	47.0	164.6	190	17
getopt.c	83.6	42.0	153.8	83	9
lalr.c	60.8	33.5	55.3	143	21

4.2 考察

表 1，2から，SC_1 と SC_2 はほぼ比例し，SE は SC より敏感に値が変わる．RTL 数と $v(G)$ は高い相関があるが，行数は他の値と殆ど関連しないことが分かった．

GCC は，計算機によって異なる RTL を出力するため，マシンに独立に評価を行うためには，仮想マシンをターゲットとしたマシン定義をする必要がある．この際，最初に生成される RTL は C 言語より低機能な命令列であるため，最適化後に C 言語の標準的な 1 文が 1 命令になるようなマシン定義が望ましい．

5. おわりに

C 言語プログラムの複雑性評価を GCC の中間コードを用いて実現した．今後は，最適なマシン定義を行い，本稿で述べた複雑性評価方式を改善するために，C 言語に最適な係数を見つけ，各評価尺度に重み付けを行った新しい評価尺度を検討する予定である．

最後に日頃御指導頂く KDD 研究所浦野所長，眞家次長に感謝する．

参考文献

[1] Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, SE-2(4), Dec 1976.

[2] M Halstead, "Programming Effort", Elements of Software Science, 1988.

[3] Ken S. Lew, Tharam S. Dillon and Kevin E. Forward, "Software Complexity and Its Impact on Software Reliability", IEEE Tr on Software Eng, pp. 1645-55, Nov 1988.

[4] "GNU C Compiler online (Tex)info Documentation for GCC 2.1", Free Software Foundation, 1992.

[5] Donald E. Knuth, "An Empirical Study of Fortran Programs", Software-Practice and Engineering, vol. 1, Nol2, 1971.

[6] Elaine J. Weyuker, "Evaluating Software Complexity Measures", IEEE Tr on Software Eng, pp. 1357-65, Sept 1988.