Vol. 26 No. 2 情報処理学会論文誌 Mar. 1985

形式文法によるプログラム複雑度の特徴づけ†

有澤 誠††

プログラムの複雑度の尺度として，Halstead 尺度や McCabe 尺度があるが，ここでは McCabe のような制御の流れに関する尺度を考察する．McCabe 尺度は，プログラムを制御フローグラフに抽象化したものを基礎にした尺度であるが，この抽象化の段階で失われる情報のために限界がある．本稿で提案する方法では，プログラムを形式文法の形に抽象化し，その文法の Chomsky 階層の水準や，非終端記号の数，生成規則の数などによって，プログラムの複雑度を特徴づける．とくに，再帰的な生成規則の数は，プログラム実行回数を制御するパラメータの数に相当し，プログラムの実行回数頻度統計（プロフィール）のパラメータ数にあたる複雑度の一つの指標である．本稿では，ソーティング，バックトラッキング，およびリストのマーキングの例について，流れ図の形のプログラム（アルゴリズム）から制御フローグラフを作り，流れ図のフロー解析結果を反映させた形式文法を対応させるようすを示し，提案している方法を具体的に説明する．またこれらの例を通して，この方法の現時点での限界を述べ，それらを克服するための見通しについて触れる．

1. まえがき

プログラムの複雑度を客観的定量的に測定することは，ソフトウェア開発工程に品質管理手法を適用する場合に重要である．プログラムの複雑度は，Boehm[1]が掲げたソフトウェア保守特性のうちで，理解しやすさ (understandability) および改修しやすさ (modifiability) と密接な関係があるからである．

これまで提案されてきたプログラムの複雑度の尺度のなかでは，Halstead によるソフトウェア科学[2]と，McCabe によるサイクロマティック数[3]とが，比較的有望視され，研究や実験が行われている[4]．

Halstead の尺度は，プログラム中のオペレータ種類数 (η_1)，オペランド種類数 (η_2)，オペレータ出現総数 (N_1)，およびオペランド出現総数 (N_2) を用いて，

$$E=\frac{\eta_1 N_2(N_1+N_2)\log_2(\eta_1+\eta_2)}{2\eta_2}$$

をプログラムの複雑度の尺度とする．データに関する情報を反映している利点はあるが，静的な尺度であるところに限界がある．

McCabe の尺度は，プログラムを制御フローグラフに変換し，そのグラフ中の始点から終点までの独立な径路数を複雑度の尺度とする．グラフ中の節点数 (n)，枝数 (e)，および連結成分数 (c) を用いて，

$$v=e-n+2c$$

で求めることができる．プログラムの制御の流れに関する情報を反映している利点はあるが，データに関する情報は含まれないところに限界がある．McCabe 尺度については，分枝点での論理式を分解することや，構造性を欠くパターン数を加味するなどの改良案も提案されているが[5)~7)]，まだ満足すべき尺度とはいえない．

本稿では，McCabe 尺度を拡張して，制御フローグラフ上の可能な径路に関する解析結果を，プログラム複雑度の特徴づけに用いることを提案する．プログラムの実行可能な径路を，形式文法[8]によって表現する．プログラムに対応する形式文法が，Chomsky 階層のどの水準にあるか，その文法の非終端記号数や生成規則数がいくつか，などの量を，プログラム複雑度の特徴づけに用いる．

以下の節では，例題を用いてこの手法を説明し，あわせてこの手法の利点と限界とを論じていく．以下の諸例では，プログラムを流れ図で表現したものから出発し，プログラミング言語で記述したものは隠してしまっている．したがって本稿の範囲では，プログラムとアルゴリズムとを，比較的近い意味に扱っている．

2. 形式文法によるプログラム制御フローの表現

プログラムを流れ図で表現したものを考える．例として**図1**の隣接ソートをとりあげる*．このアルゴリズムは文献[9]に引用されている．この流れ図を，制御フローグラフに抽象化したものが**図2**である．グラフの節点には，A, B, …と名前をつけ，始点と終点はそれぞれSとEと名前をつける．アルゴリズムの処理本

† Formal Grammar Characterization of Program Complexity by MAKOTO ARISAWA (Department of Computer Science, Faculty of Engineering, Yamanashi University).

†† 山梨大学工学部計算機科学科

* 流れ図中に，節点や枝が実行される回数をパラメータを含む式の形で円（長円）にかこんで示してある．

体は，グラフの枝に吸収されている．そこで枝には，1, 2, …と名前をつける．

このグラフのSからEへの独立な径路は4であり，McCabe 尺度 $v=4$ が得られる．このグラフ上の実行

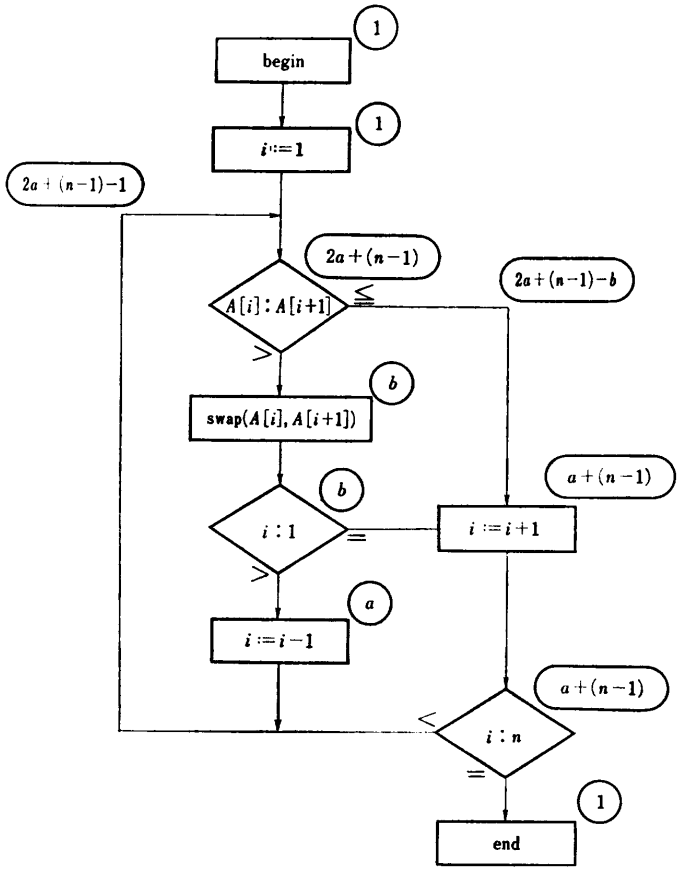

図 1 隣接ソートのアルゴリズム (Maurer によるもの)
Fig. 1 Adjacent exchange sorting algorithm by Maurer.

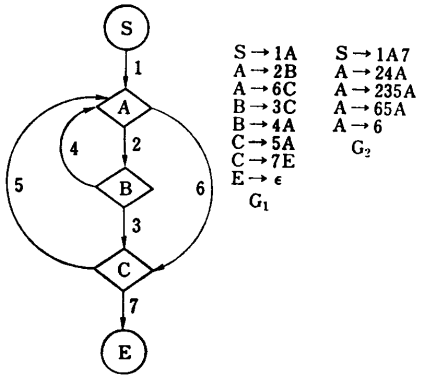

図 2 ソートアルゴリズムの制御フローグラフと径路記号列生成の文法
Fig. 2 Control flow graph for the sorting algorithm and formal grammars to generate symbolic paths.

可能な径路を，枝につけた名前の列で表す．たとえば1237は一つの径路であり，12467は別の径路である．このように径路を枝の名前の列で表したものを径路記号列とよぶ．

さて与えられたグラフについて，径路記号列をすべて生成する形式文法を考える．終端記号として枝の名前，非終端記号として節点の名前をとると，さきのグラフでは G_1 のような生成規則が得られる．以下，形式文法は生成規則の集合の形で表すが，終端記号集合，非終端記号集合，開始記号が何にあたるかは自明であろう．

G_1 は正規文法で，非終端記号数は節点数に，終端記号数は枝数にそれぞれ等しく，McCabe 尺度の算出と同じ水準の情報が反映されている．一般に，プログラムの制御フローグラフの形に抽象化したものから，すべての径路記号列を生成するような正規文法を作ることができる．すなわち，節点Pから枝kが節点Qにむかっているとき，P→kQ を生成規則とすればよい．すべての枝についてこのように生成規則を作り，最後に終点名Eについて E→ϵ をつけ加えれば，それが求める正規文法である．

さきの文法 G_1 は，非終端記号Eが右辺に現れる部分をϵでおきかえてEを除いたあと，非終端記号CとBについても，順にその右辺をCやB自身の位置に埋めこむと，G_2 のような片側線型文法が得られる*．非終端記号Aと開始記号Sは，自己埋めこみでは除去できない．一般に任意の文脈自由文法は，開始記号と再帰的な規則をもつ非終端記号だけを残し，他の非終端記号は埋めこみによって除去できることが知られている[10]．

文法 G_2 を図1の流れ図に対応させてみると，径路記号列を生成するときに，個々の生成規則を施す回数にプログラムからの意味づけができる．ソートされる配列の大きさを n，その配列の初期状態のインバージョン（大小順序逆転）数[11]を b，同じく初期状態の左から右への最小値の個数を $a+1$ とする．このとき，A→24A を施す回数は $b-a$，A→235A を施す回数は

* 開始記号の部分だけは片側線型でないが，片側線型に直すことは容易であり，また G_2 の形のほうが簡潔で扱いやすいので，以下開始記号については制約をゆるめたものを片側線型とよんでいく．

a, A→65A を施す回数は $b+n-2$ であることが計算できる．この計算には，流れ図のある節点に入る枝を通る回数の合計とそこから出る枝を通る回数が，始点終点以外の節点では等しくなること，および i を増す回数と i を減らす回数の差が $n-1$ であること，を利用している．

図1の流れ図に対応するプログラムは，片側線型文法の3通りの再帰的な生成規則を施す回数として，3個のパラメータで特徴づけができた．すなわち，文法 G_2 はプログラムの各部分を実行する回数と，径路記号列の構造とを示している．このプログラムは，非終端記号2種，再帰的非終端記号1種，生成規則5個，再帰的生成規則3個の，片側線型文法で特徴づけられることがわかった．文法の Chomsky 階層の水準からみても，終端記号や生成規則の数からみても，複雑度は低いといえる．

2番目の例として**図3**のバックトラッキングをとりあげる．このアルゴリズムは，文献[12]に引用されており，あらかじめ定めた深さ n を限度に縦探索を行い，そのときにバックトラッキングをするものである．図3の流れ図を，制御フローグラフに抽象化したものが**図4**である．前の例と同様に正規文法を対応づけすると G_3 のようになる．

文法 G_3 は，前の例の G_1 とは異なって，図4の径路記号列を生成するものの，ただちにそれは図3の流れ図上での径路とは対応しない．たとえば15657という径路記号列は図4のグラフでは可能な径路であるが，図3の流れ図ではありえない*．すなわち，図3を図4に抽象化した段階で，プログラムの実行制御に関する情報が一部分消えてしまっている．McCabe の尺度は，この図4のグラフから $v=4$ を与えているが，この値を図3の流れ図で示されるプログラムの複雑度として用いるところに，McCabe 尺度の粗さの一面がある．

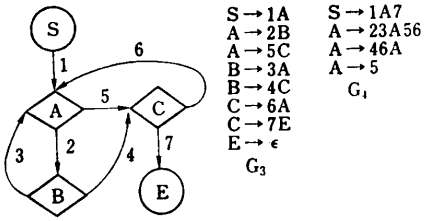

図4 バックトラッキング・アルゴリズムの制御フローグラフと径路記号列生成の文法

Fig. 4 Control flow graph for the backtracking algorithm and formal grammars to generate symbolic paths.

図3の流れ図で，探索した項目の個数が a，そのうち解として印刷した項目の個数が b である．この記号パラメータに関する実行回数の関係式に注意して，径路記号列を生成するような文法を求めると，G_4 のような線型文法になる．ここで，A→23A56 という生成規則は片側線型でなく，Chomsky 階層の水準が一段複雑になる．この生成規則によって，23という径路の出現回数（$k:=k+1$ の実行を含む部分）が，56という径路出現回数（$k:=k-1$ の実行を含む部分）と等しいという情報を表現している．A→23A56 を施す回数が a，A→46A を施す回数が b である．すなわち，実行回数を制御するパラメータ数，いいかえると再帰的な生成規則の種類は2である．G_4 は，非終端記号2種，再帰的非終端記号1種，生成規則4個，再帰的生成規則2個の，線型文法である．

図1のプログラムと図2のプログラムの複雑度を比べると，形式文法の Chom-

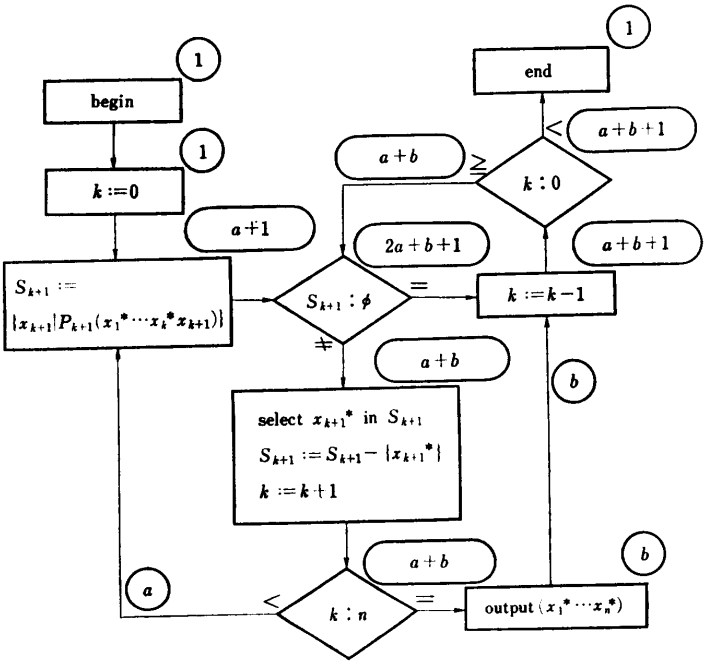

図3 深さ固定（n）のバックトラッキング・アルゴリズム（Knuth によるもの）

Fig. 3 Knuth's backtracking algorithm with fixed depth limit n.

* 15657という径路は，$k=0$ の状態で $S_1:=\phi$ を実行し，$k:=k-1$ のあと $k:0$ の判定で $k\geqq 0$ の側へ分岐してしまうことに相当している．

sky 階層の水準では前者のほうが単純であり，実行回数を制御するパラメータの面では後者のほうが単純である．この対比は，McCabe 尺度を用いて，どちらも $v=4$ で複雑度は同じ，と割り切ってしまうことのできない論点をもっている．実行回数を制御するパラメータの数の差異は，たとえばプロファイラ[13]を用いてプログラムの実行頻度統計を作ったとき，その頻度がいくつの独立な値が組み合わされて出てきたかの差異にあたる．この独立なパラメータ数は，プログラムの仕様から定まることが多い．しかしたとえばソーティングの例をとると，与えた配列の要素数は，おそらくソーティングのアルゴリズムによらず，実行回数を制御するパラメータの一つになるであろうが，左から右への最小値の個数は，アルゴリズムによっては，このパラメータにならないであろう*．その代りに，別の量が実行回数を制御するパラメータになり，その数もアルゴリズムによって変わるであろう．したがって，独立なパラメータ数の一部はプログラムに依存し，一部はアルゴリズムに依存する．

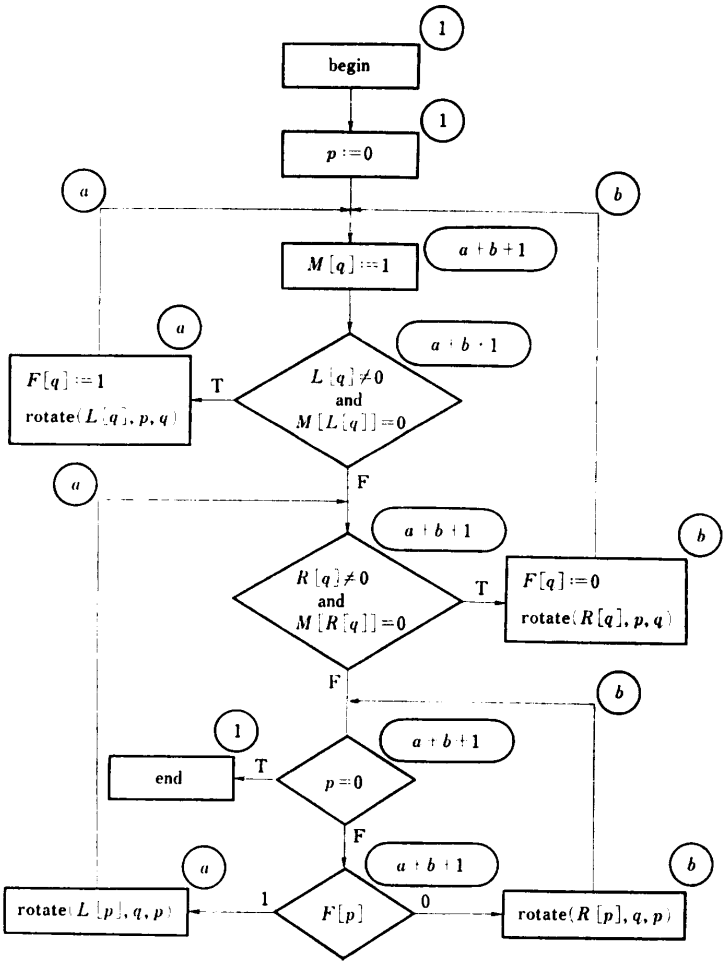

図 5 ポインタ反転のリストマーキングアルゴリズム (Schorr-Waite によるもの)

Fig. 5 Schorr-Waite's pointer reversal list marking algorithm.

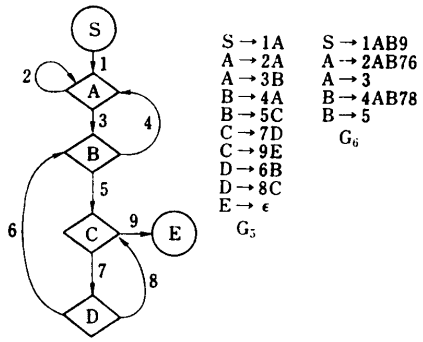

図 6 マーキングアルゴリズムの制御フローグラフと径路記号列生成の文法

Fig. 6 Control flow graph for the marking Algorithm and formal grammars to generate symbolic paths.

本稿で提案している形式文法によるプログラム複雑度の特徴づけでは，この独立なパラメータ数が，生成規則を施す回数に自由度があるような生成規則の数に対応している．

3番目の例として**図5**のリストマーキングをとりあげる．このアルゴリズムは，文献[11]に引用されているものを，rotate 操作**を用いて書き直したものである．図5の流れ図を，制御フローグラフに抽象化したものが**図6**である．ここでも機械的に正規文法を対応づけすると，G_5 のようになる．

文法 G_5 もまた，図6の径路記号を生成するものの，その一部の径路記号列は図5の流れ図上での径路には対応しない．たとえば 135789 という径路記号列は図6のグラフでは可能な径路であるが，図5の流れ図ではありえない***．ここでも，図5を図6に抽象化した段階

* たとえばラディックスソートのアルゴリズムなどの場合．

** rotate(xyz) とは $x \leftarrow y \leftarrow z \leftarrow x$ のように順送りにいれかえる操作を表す．

*** 135789 という径路は $p=0$ を実行し，次の二つの判定をFで通ったあと $p=0$ の判定をFで通ることに相当している．

で，情報が一部消えている．図6のグラフに対して McCabe が与える複雑度は $v=5$ である．

図5の流れ図には4個所に rotate 操作が現れる．左ポインタをたどる，右ポイントをたどる，左ポインタへもどる，右ポインタへもどる，にそれぞれ対応している．そこで実行回数を制御するパラメータとしては，左ポインタをたどる回数(a)と，右ポインタをたどる回数(b)の2種類を選べば，たどった回数ともどった回数が等しいことから，すべての実行回数が定まる．すなわち，図6のグラフで2および6の枝の実行回数が a，4および8の実行回数が b である．このことに注意して文法を組みたて直すと，G_6 のような文脈自由文法が得られる．Chomsky 階層では，片側線型，線型に次ぐ水準である．

G_6 で，A→2AB76 を施す回数が左ポインタをたどる回数，B→4AB78 を施す回数が右ポインタをたどる回数にあたる．非終端記号は3種，再帰的非終端記号2種，生成規則5個，再帰的生成規則2個の文脈自由文法である．さきの二つの例に比べて，非終端記号の種類が増え，文法の水準も複雑になった．しかし実行回数を制御するパラメータ数では，最初のソートの例よりも少なくなっており，その面ではむしろマーキングの例のほうが単純である．すなわち，文法の水準は構造面からみた複雑さを特徴づけており，文法のもつ生成規則を施す自由度の種類（実行回数を制御するパラメータ数）は構造面とは別の次元の複雑さを特徴づけている．

構造面の複雑さは，どちらかといえば静的な複雑さであり，実行回数の制御パラメータは，それよりは動的な複雑さである．ただ，プログラムのプロフィールについて，それがプログラムの各部分の実行回数の累計であり，動的特性を静的に表したところに限界があるという議論は，ここでも成りたつ．たとえば文法 G_6 についてみると，A→2AB76 および B→4AB78 を施す回数は，それぞれ a と b である．しかし $a=2$, $b=2$ を与えても，これらの生成規則を施す順序にまだ自由度が残っており，順序を変えると別の径路記号列が生成できる．**図7**に示す2通りのリスト構造が，$a=2$, $b=2$ として生成規則を施す順序を変えた場合に対応する．どちらも左ポインタを2回右ポインタを2回たどっているが，たどったポインタのあともどりと，次のポインタをたどるときの順序が異なっている．この差異はプロフィール上には現れない．形式文法では生成規則を施す順序の自由度の形で暗黙的に情報が含まれているわけである．

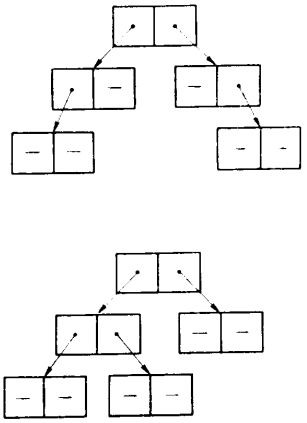

図7　マーキングで $a=2$, $b=2$ に対応する2種類のリスト構造

Fig. 7　Two different list structures which correspond $a=2, b=2$ case in the marking algorithm.

これまで3通りの例をあげて，制御フローグラフの径路記号列を生成する形式文法を求め，その文法によってプログラムの複雑度を特徴づける方法を示してきた．制御フローグラフに抽象化されたものから複雑度を定める McCabe 尺度と異なり，形式文法を求める際にはもとの流れ図の実行回数を制御するパラメータのもつ情報をとりいれている点が，この方法の特色である．

3. 形式文法による特徴づけの考察

流れ図の実行状況を，径路記号列の形に抽象化することは，Halstead の方法のような個々の演算やデータに関する複雑度を無視し，McCabe の方法と同様に制御の流れのもつ複雑度にのみ注目したことにあたる．そのうえで，もとの流れ図のもつ意味論をある程度反映させた形式文法の形にさらに抽象化し，この形式文法でプログラムの複雑度を特徴づけすることをねらっている．

前章の例でも示したように，流れ図のもつ意味論を無視すれば，制御フローグラフから必ず正規文法が得られる．この文法，あるいはこれに文献[10]の方法を適用して得られた片側線型文法に，McCabe 尺度のもつ情報がそのまま対応している．もとの流れ図のフロー解析を行って，径路記号列に関する制約を調べ，それを反映させて Chomsky 階層の水準がより高い形式文法を求めることができれば，プログラムの複雑度がよりていねいに特徴づけできる．

形式文法の水準は，プログラムの構造の複雑度を示

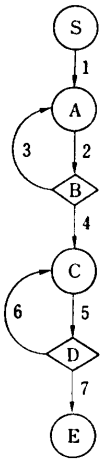

図 8 二つのループがある制御フローグラフ
Fig. 8 Control flow graph with two loops.

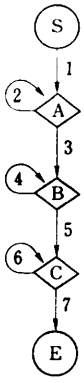

図 9 三つのループがある制御フローグラフ
Fig. 9 Control flow graph with three loops.

す指標である．たとえば**図 8** の二つのループをもつ制御フローグラフでは，23 という径路記号のループと 56 という径路記号のループが独立なら，対応する文法は片側線型である．しかしこの二つのループをまわる回数が同じであるという情報が加わると，対応する文法は線型になる*．このとき，文法の水準が一つあがり，実行回数を制御するパラメータは 2 から 1 に減る．同様に**図 9** の二つのループをもつ制御フローグラフでは，それぞれのループが独立かどうかで，3 通りの異なる水準の形式文法が対応する．三つのループがすべて独立なら片側線型文法，どれか二つのループをまわる回数が同じなら線型文法である．三つのループをすべて同じ回数まわるときには，文脈依存文法を用いなければならなくなる．しかしその分，実行回数を制御するパラメータ数は順に減っていき，文法の水準が上がっても，プログラムの複雑度としては，別の面で単純化がすすむ．なお，三つのループの独立性については，あるループをまわる回数が別のループをまわる回数のちょうど 2 倍であるとか，別のループをまわる

* S→1A7, A→23A65, A→245.

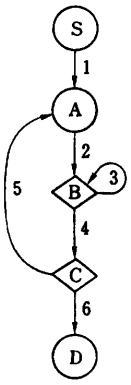

図 10 ネストしたループがある制御フローグラフ
Fig. 10 Control flow graph with nested loops.

回数よりちょうど 1 回多いなどの関係であっても，形式文法の水準は同様に上がっていく．

図 10 はネストしたループをもつ制御フローグラフである．内側のループが外側のループと独立であれば，片側線型文法が対応する．しかし内側のループと外側のループが独立でなく，内側のループをまわる回数が外側のループに依存する場合には，文脈依存文法が対応することになる．

片側線型，線型，文脈自由という階層の差は，プログラムの構造的な複雑度を特徴づけするために，直観的に理解しやすい．しかし文脈依存文法に至ると，構造的な複雑度が文脈自由文法の範囲を越えたという情報にとどまる点が，本稿で提案している文法の現時点での限界の一つである．文脈依存文法については，さらにその中をいくつかの水準に細分したモデルをとりいれるよう，拡張することを検討中である．

プログラムの実行回数を制御するパラメータ数は，プログラムの複雑度を特徴づけるもう一つのよい指標である．この値が小さい場合，プログラムの構造が複雑であっても，その流れを追うことはそれほど困難ではない．逆にこの値が大きいと，プログラムの各部分を実行する順序の自由度が大きくなり，プログラムの流れを理解することは容易でなくなる．

本稿で述べている方法のうちで，制御フローグラフから機械的に得られた正規文法に手を加え，もとの流れ図のフロー解析の情報を利用してより水準の高い形式文法を求める部分は，McCabe 尺度や Halstead 尺度のように機械的には行えず，人間の知的活動を必要とする．この形式文法を求める過程で，プログラムに対する理解がすすむので，この複雑度の特徴づけをプログラム評価の工程にとりいれることには意義があるものの，できれば機械的あるいはそれに近い形で形式

文法が求まることが望ましい．この手法の現時点でのもう一つの限界はこの点である．フロー解析を行うトゥールはすでにいろいろあるので[14]，そのようなトゥールを組みこんで，対話型式で形式文法を求めるシステムを検討中である．部分的に文法推論の手法をとりいれることを含めて，形式文法を求める過程に費す人間側のエネルギを少なくすることを考慮中である．

プログラムの複雑度を，一つのスカラ値で表現した場合には，複雑度のある側面だけしか反映できない．プログラムを，その複雑度を保持したまま抽象化したモデルに変換できれば，そのモデルは複雑度の多様性を残したまま抽象化したものであることから，複数のプログラムの複雑度の比較を行ったり，複雑度に関する評価を行う場合に使用できる．このようなモデルとして，本稿では形式文法をとりあげ，形式文法によってプログラムの複雑度を特徴づけすることについて述べた．さきに触れた現在検討中の2点についても，具体化した結果が得られた段階で報告する予定である．

謝辞 本研究について助言をいただいた，Connecticut 大学 EECS 学科教授 T. L. Booth 教授に感謝する．

参考文献

1) Boehm, B. W. et al.: Quantitative Evaluation of Software Quality, Proc. 2nd Int. Conf. Soft. Eng., pp. 592-605 (1976).
2) Halstead, M. H.: *Elements of Software Science*, Elsevier (1977).
3) McCabe, T. J.: A Complexity Measure, *IEEE Trans.*, SE 2-4, pp. 308-320 (1976).
4) Curtis, B. et al.: Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics, *IEEE Trans.*, SE 5-2, pp. 96-104 (1979).
5) Myers, G. J.: An Extension to the Cyclomatic Measure of Program Complexity, *SIGPLAN Notices*, 12-10, pp. 61-64 (1977).
6) Elshoff, J. L. and Marcotty, M.: On the Use of the Cyclomatic Number to Measure Program Complexity, *SIGPLAN Notices*, 13-12, pp. 29-40 (1978).
7) Oulsnam, G.: Cyclomatic Numbers Do Not Measure Complexty of Unstructured Programs, *Inf. Process. Lett.*, 9-5, pp. 207-211 (1979).
8) Hopcroft, J. E. and Ullman, J. D.: *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading (1969) (野崎, 木村訳: サイエンス社).
9) Maurer, W. D.: Comments on Social Process and Proofs, *Comm. ACM*, 22-11, pp. 625-629 (1979).
10) 有沢, 鳥居：単純句構造言語の自己埋込み標準形について，電子通信学会論文誌，57 D-5, pp. 324-325 (1974).
11) Knuth, D. E.: *The Art of Computer Programming*, Vol. 1, *Fundamental Algorithms*, Addison-Wesley, Reading (1968, 1973) (広瀬, 米田, 筧訳: サイエンス社).
12) Knuth, D. E.: Estimating the Efficiency of Backtracking Programs, *Math. of Computation*, 29, pp. 121-136 (1975).
13) 牛島：Fortran プログラミング・ツール，産業図書 (1980).
14) Wetmore, T. T., Ⅳ: The Performance Compiler, A Tool for Software Design, Univ. Connecticut Technical Report (1980).

(昭和59年4月20日受付)
(昭和59年9月20日採録)