コンパイラのテスト網羅性判定ツール：C-GRAM

上原憲二、堀川博史、大川勉、高野彰、春原猛
三菱電機㈱　情報電子研究所

1. はじめに

コンパイラは重要基本ソフトウェアの一つである。しかしその作成、テストはコンパイラ構築等に豊富な経験を持つ者によって行われ、テストの完全さは担当者の経験や勘に頼る部分が大であり、不注意によるテストデータの漏れなどの問題がある。本報告ではこのような問題を解決するために、コンパイラのテストデータ（ソースプログラム）の網羅性判定の一つの方法とそれを支援する言語独立なツールを提案し、またこのツールの試使用による評価を示す。この方法は文脈自由文法で定義される言語の構文規則全体が、すべてのテストデータを構文解析する際に適用されたか否かという網羅性（構文的網羅性と呼ぶ）基準によるものである。このテスト網羅性判定法を支援するツールC-GRAM（Coverage checker based on GRAMmar）は直構文解析を用いて構成される。従って、いかなる言語についても文脈自由文法に基づく構文規則を作成すればC-GRAMを使用できる。

2. 構文規則の定義

構文規則は図１に示す拡張Backus-Naur形式(EBNF)により定義される。(以下では図１のEBNFを標準形と呼ぶ。)生成規則の右辺の式は繰返し部を含むことができる。(式)+ は１回以上の繰返しを示し、(式)* は０回以上の繰返しを示す。一般にEBNFはグループ化部｛式｝やオプショナル部［式］のような簡便記法を含んでいる。しかし構文的網羅性判定の観点からそれらを展開した形（図２参照）の方が網羅性判定基準の定義のためには適切であるので、標準形は簡便記法を含んでいない。

```
構文規則 ::= ( 生成規則 )+ .
生成規則 ::= 非終端記号 "::=" 式 "." .
式       ::= 項 ( "|" 項 )* .
項       ::= ( 因子 )+ .
因子     ::= 非終端記号 | 終端記号 |
             "(" 式 ")+" | "(" 式 ")*" .
非終端記号 ::= 識別子 .
終端記号 ::= """ 文字列 """ .
```

図１．　EBNFの構文

```
A ::= {"X"|"Y"}B{"U"|"V"}.
B ::= "P"["Q"]"R".
    ⇓ 展開
A ::= "X"B"U"|"Y"B"U"|"X"B"V"|"Y"B"V".
B ::= "P""Q""R"|"P""R".
```

図２．　簡便記法の展開

3. 判定基準

構文的網羅性判定のため標準形の生成規則に対して次の尺度を定義する。それらは生成規則の各部分(以下ではセグメントと呼ぶ)がすべてのテストデータを書くのに何度使用されたかを示すものである。

n_p：生成規則 p の適用回数

n_t：項 t の適用回数

$n_{\alpha+1}, n_{\alpha+2}$：$(\alpha)+$ がそれぞれ１回又は２回以上の繰返しで適用された回数

$n_{\alpha*0}, n_{\alpha*1}, n_{\alpha*2}$：$(\alpha)*$ がそれぞれ０回，１回又は２回以上の繰返しで適用された回数

これらを「０階の尺度」と呼ぶ。更に詳細な判定のためにより高階の尺度を以下のように定義する。生成規則について、その右辺に含まれる非終端記号を下位の生成規則、すなわち、その非終端記号の生成規則の右辺で置き換え、標準形に展開する。このような置き換えと展開((下位の)生成規則の組合せと呼ぶ)により得られた生成規則に対して上記と同様の尺度を定義する。この尺度を「１階の尺度」と呼び上に添字付けして n_p^1, n_t^1 などと記す。(０階の尺度は必要な場合 n_p^0, n_t^0 などと記す。)同様にして、２階、更に高階の尺度を定義できる。（図３参照）

```
A ::= B C。
B ::= D|"X"。
C ::= "U"|"V"。
D ::= "Y"|"Z"。
   ⇓ BとCの置き換え
A ::= {D|"X"}{"U"|"V"}。
   ⇓ 展開
A ::= D"U"|D"V"|"X""U"|"X""V"。 ――――①
   ⇓ Dの置き換え
A ::= {"Y"|"Z"}"U"|{"Y"|"Z"}"V"|
      "X""U"|"X""V"。
   ⇓ 展開
A ::= "Y""U"|"Z""U"|"Y""V"|"Z""V"|―②
      "X""U"|"X""V"。
```

①,②に対して、それぞれ1階、2階の尺度が定義される。

図3. 生成規則の組合せ

上記の尺度に基づいて次の構文的網羅性判定基準を定義する。

[基準1]

もし η_x^k ($k=0.1.2\cdots$, $x=p.t, d+1, d+2, d*0, d*1, d*2$) の値が0ならば、対応するセグメントはテストされていない。

[基準2]

もし η_x^k ($k=0.1.2\cdots$, $x=p,t,d+1,d+2,d*0,d*1,d*2$) の値がある標準値より小さいならば、対応するセグメントに対するテストデータは十分でない。そのような標準値の一つとして、上述した生成規則の組合せの大きさを示す生成規則の複雑度（付録参照）を用いることができる。

4. 判定手順

生成規則についてデータ η_x^k（尺度）及び C_x^k（複雑度）をツールを用いて収集し、前章で述べた基準に従ってテストデータの構文的網羅性を判定する。判定をより効果的なものにするため、次の手順に従うことが望ましい。

[ステップ1：構文規則の調整]

まず、対象言語の構文規則を定義しなければならない。一般に、言語仕様書ではある種のEBNFを用いて構文規則が定義されている。この構文規則を標準形に書きかえればよいであろう。しかし、言語仕様書での構文定義では使用者にわかり易くするための工夫がされているが、そのような構文定義はテスト網羅性判定のためには必ずしも適当であるとは言えない。そのため、以下に示すような構文規則の調整をすべきである。

- 階数kが1又は2であっても生成規則によっては、その組合せの大きさすなわち複雑度 C_p^k が大きすぎて高階の η_x^k のデータが事実上収集不可能になることがある。この場合はいくつかの中間的な生成規則を導入することにより複雑度を減ずる必要がある。図4の例では、X、Y、U及びVの中間的生成規則の導入による調整で、Aの生成規則の1階の複雑度 C_{pA}^1 は、①では400から②では4に減少している。
- 下位の生成規則の組合せを行うことにより、構文上正しいが意味上正しくない項が作られる場合がある。例えば、理解を容易にするために簡便記法を用いており、言語仕様書中に意味上正しくない組合せを示すコメントが付けられた生成規則があるであろう。このような組合せはできるだけ取除く必要がある。何故ならばそれらは未テストのセグメントと判定されたり、また複雑度の値が大きくなりすぎる原因でもあるからである。すなわち、網羅性判定に雑音を入れる。
- 一般に、非終端記号はいくつかの生成規則の右辺で参照される。これは、それら参照間に構文上の違いはないが意味上の違いのある表われと考えられる。もし、重要な意味上の違いがあるなら

```
A ::= B C。
B ::= "X1"|"X2"|...|"X10"|"Y1"|"Y2"|...|"Y10"。 }①
C ::= "U1"|"U2"|...|"U10"|"V1"|"V2"|...|"V10"。 }
   ⇓
A ::= B C。
B ::= X | Y。
C ::= U | V。
X ::= "X1"|"X2"|...|"X10"。
Y ::= "Y1"|"Y2"|...|"Y10"。
U ::= "U1"|"U2"|...|"U10"。
V ::= "V1"|"V2"|...|"V10"。
```

②

図4. 中間的生成規則の導入

ば、その非終端記号の生成規則及びいくつかのそれの下位の生成規則からなる集合に対するデータは、各参照ごとに収集されることが望ましい。このために、その生成規則といくつかのそれの下位の生成規則の集合を複製することによって、各参照ごとに専用の生成規則の集合を作ることが必要である。例えば図5では、代入文の右辺の式の値は単に格納されるだけであるが、配列変数参照における添字式の値は番地を得るのに用いられるので、各々別々にデータを収集するために専用の式、項及び因子の生成規則を複製している。このような複製処置は、参照間で重要な意味上の違いを持つすべての参照に対して行う必要がある。

```
ASSIGN      ::= VARIABLE "=" EXPRESSION ";".

INDEXEDVAR  ::= VARID "(" EXPRESSION ")".

EXPRESSION  ::= TERM ( ADDOP TERM )*.
TERM        ::= FACTOR ( MULOP FACTOR )*.
FACTOR      ::= VARIABLE | CONSTANT | "(" EXPRESSION ")".
                ⇓
ASSIGN      ::= VARIABLE "=" AEXP ";".
AEXP        ::= ATRM ( ADDOP ATRM )*.
ATRM        ::= AFCT ( MULOP AFCT )*.
AFCT        ::= VARIABLE | CONSTANT | "(" AEXP ")".

INDEXEDVAR  ::= VARID "(" IEXP ")".
IEXP        ::= ITRM ( ADDOP ITRM )*.
ITRM        ::= IFCT ( MULOP IFCT )*.
IFCT        ::= VARIABLE | CONSTANT | "(" IEXP ")".

EXPRESSION  ::= TERM ( ADDOP TERM )*.
TERM        ::= FACTOR ( MULOP FACTOR )*.
FACTOR      ::= VARIABLE | CONSTANT | "(" EXPRESSION ")".
```

図5. 生成規則の複製

[ステップ2：導出木生成]

ステップ1で作成された構文規則に従って、すべての構文上正しいテストデータを直構文解析してそれらの導出木を生成する。

[ステップ3：データ収集と網羅性判定]

まず、すべての導出木をトラバースして、0階の尺度のデータD_0^αをすべての生成規則、項及び繰返し部について収集する。収集したデータを3章で述べた基準に照らし構文的網羅性を判定する。次により詳細な判定が必要と思われる生成規則について高階の尺度のデータD_k^α $(k>0)$を収集し、基準に従って判定する。これは、一般に生成規則αがその右辺に非終端記号を含めば、階数kの増分に対しk階の複雑度C_k^αの増大は非常に大きく、事実上D_k^αの収集が不可能になるため、すべての生成規則についてはD_k^αを収集することはできないので選択的に行う。従って、詳細な判定が必要なものに対しては、あらかじめ適切な複雑度の低減を行っておく必要がある（ステップ1参照）。

5. C-GRAMの構成

コンパイラのテストデータの構文的網羅性判定法を支援するツールC-GRAMを構成するモジュールとそれらの機能の概要を以下に述べる（図6参照）。

(1) 構文規則調整モジュール

構文規則調整モジュールは4章で示した判定手順のステップ1を支援する。入力の構文規則は簡便記法を含んでいてもよい。それらはこのモジュールにより展開されて標準形の構文規則が作られる。また、非終端記号、終端記号の相互参照リストや複雑度のデータC_k^αが出力される。構文規則の調整はこのような情報をもとに対話形式で行われる。

(2) テーブル生成モジュール

調整された構文規則を入力し、直構文解析に必要なテーブル類が生成される。それらは、終端記号表、非終端記号表、構文表及び選択表と呼ばれるものである。

(3) 字句解析モジュール

字句解析モジュールはテストデータを終端記号表に従って、終端記号列に変換する。C-GRAMでは字句単位を終端記号と考

える。従って、テストデータ中の識別子、定数は終端記号として扱われる。

(4) 直構文解析モジュール

直構文解析モジュールは、非終端記号表、構文表及び選択表による表駆動方式で、終端記号列を直構文解析しその導出木を生成する。

モジュール(2)～(4)はステップ2を支援する。(2)は調整された構文規則に対して一度だけ適用される。(3)と(4)はすべての構文上正しいテストデータに対して適用される。

(5) データ収集モジュール

データ収集モジュールはステップ3を支援する。すなわち、すべての導出木をトラバースしてデータ次を収集し、それらを報告する。データ次は常に収集される。他方、データ次($k>0$)はユーザの指定した生成規則、階数について収集される。この際のトラバースは導出木全体ではなく指定された生成規則に対応する部分木に対してのみ行われる。

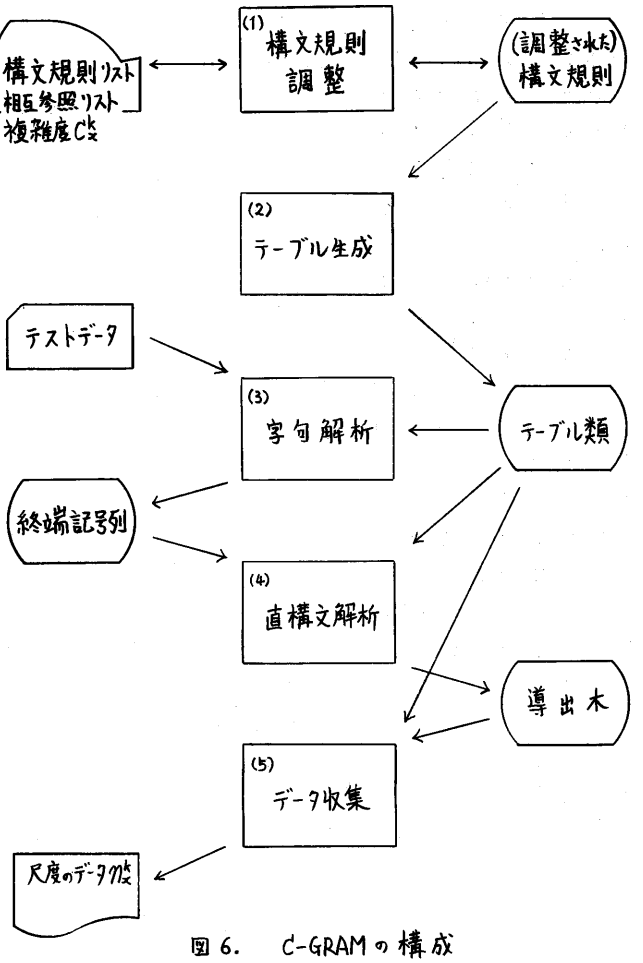

図6. C-GRAMの構成

6. C-GRAMの評価

C-GRAMの有効性を評価するために、テーブル生成モジュール、字句解析モジュール、直構文解析モジュール[1],[2]及びデータ収集モジュールから構成されるプロトタイプのC-GRAMを実現した。[3],[4] 各モジュールは主にPL/Iで書かれ、一部アセンブリ言語で書かれた。各々のサイズを表1に示す。次に我々は、実際に開発されたコンパイラの総合テスト段階で作成、使用されたテストデータに対してこのC-GRAMを適用した。(但し、このプロトタイプでは字句解析モジュールは対象言語専用のものを作成した。)

対象のコンパイラはPL/I型の言語に対するものである。構文規則を調整することによって、生成規則の数は言語仕様書で定義された数104から200に増加した。テストデータは構文上意味上正しいテス

表1. モジュールのサイズ

モジュール	言語	行数(注) (文数)
テーブル生成	PL/I	約1200 (約800)
	アセンブリ	200
字句解析	PL/I	300 (300)
直構文解析	PL/I	900 (600)
	アセンブリ	300
データ収集	PL/I	2100 (1300)
	アセンブリ	500

(注) コメント行を含む。

トケースに対して作成されたものであり、その数は645、その総行数（コメント行を含む）は約52000であった。また、それらの終端記号列中の総終端記号数は約392000であった。

テーブル生成に費したCPU時間は、MELCOM-COSMO 900Ⅱで約1分であった。すべてのテストデータについて導出木を生成し、すべての導出木をトラバースして、すべての生成規則、項及び繰返し部に対する0階の尺度のデータn_x^0を収集するのに費したCPU時間は約32分であった。また、二つの生成規則（1階の複雑度C_p^1はそれぞれ600、2310）を選んで1階の尺度のデータn_x^1を収集したが、それに費したCPU時間は約36分であった。

結局、我々はC-GRAMの有効性を示す次のような結果を得た。

- $n_p^0=0$である生成規則は5、$n_x^0=0$（$x=t, \alpha+1, \alpha+2, \alpha*0, \alpha*1$又は$\alpha*2$）であるセグメントを含む生成規則は54発見された。図7にその報告の一部を示す。これら未テストセグメントのうち三つは不注意によるテストデータの漏れによるものであり、他は意図的に省略されたものであった。コンパイラの試使用期間中に集められたコンパイラの障害報告(注)の調査によれば、二つの障害が上の未テストセグメントに関係していることが明らかになった。
- n_x^1のデータの数はそれぞれの生成規則の複雑度C_p^1の値（すなわち600と2310）だけあり、大部分のn_x^1の値は0であった。しかし、$n_x^1=0$であるセグメントに対応する障害報告が二つ存在した。

更に我々は、構文規則調整の有効性、追加テストの指針について次のような経験的な結果を得た。

- 上に述べたようにn_x^1によって、対応する障害報告の存在する未テストセグメントを二つ発見したが、そのうち一つは複雑度C_p^1を減ずる中間的生成規則の導入なしには、データn_x^1を収集することはできなかったものである。また、式に関する生成規則の集合を複製して二つの新しい生成規則の集合、一つは代入文の右辺の式用、一つは添字式用を作った。その結果、前者の式は十分にテストされているが、後者はある種の演算子について十分にはテストされていないことが判明した。
- $n_x^0=0$によって発見された未テスト部に対する追加テストを行う時は$n_x^0\neq 0$にするテストデータを作成するが、単

```
DCLST ::= "DECLARE" ( LEVEL | "" ) DCLR
3451        3451        513      2938
                      ( "," ( LEVEL | "" ) DCLR )* ";".
                        7517 4093     3424      0-2272
                                                1-214
                                                2-965

DCLR  ::= (( IDNTFYR | "*" )( DIMATR | "" ) |
17399         15943       104   1871      14176
            "(" DCLR ( "," DCLR )* ")"          )
              1352        5079    0-0
                                  1-218
                                  2-1134

          ( ATTR )*.
            25773
                  0-4555
                  1-5141
                  2-7705
LEVEL ::= ACNST.
4606      4606
```

生成規則の左辺の非終端記号の下の数字はその生成規則の適用回数（0階の尺度のデータn_p^0）を示す。'0-'、'1-'、'2-'が付けられた数字は（式）*の繰返し部の繰返し回数がそれぞれ0回、1回、2回以上による適用回数（$n_{\alpha*0}^0$、$n_{\alpha*1}^0$、$n_{\alpha*2}^0$）である。他は項の適用回数（n_t^0）を表わす。（なお、このプロトタイプではブループ化部（式）を許している。）

図7. C-GRAMの報告の一部

(注) 今回のC-GRAMの適用はコンパイラの試使用期間後に行われた。

に $n_x^0 \neq 0$ にするだけであってはならず、テストケース、テストデータのレビューをすることが重要であろう。$n_x^k = 0\ (k>0)$ の場合は、すべてのセグメントについて $n_x^k \neq 0\ (k>0)$ にするようなテストデータを作成する必要はないであろう。（前述のように $n_x^k\ (k>0)$ のデータ数は非常に多いので、それらすべてを0でなくするテストデータは冗長と思われる。）

7. おわりに

C-GRAMは言語仕様に基づいてコンパイラテストの構文的網羅性判定を支援するものである。C-GRAMのプロトタイプの実現とその試使用から、このようなツールはテストの完全さを確かめる上で、また追加テストの指針を得る上で実用的、効果的であることを確認した。C-GRAMは言語独立であり構文規則をEBNFで記述すれば任意の言語に対して容易に使用できる。（但し、プロトタイプでは字句解析モジュールも作成する必要がある。）特に、他所で作成、テストされたコンパイラの受入れ検査にC-GRAMを使用すれば効果的であろう。すなわち、受入れ検査は多くの場合人手によるテストケースのレビューと少数の受入れ検査プログラムによるテストだけであったのに対し、C-GRAMはテストデータ網羅性の定量的判定を可能にした。

今後の課題として、まず構文上正しくないテストデータに対する網羅性判定機能の追加がある。次に意味規則に基づく網羅性判定機能の実現がある。意味規則は属性文法などによって定義することが考えられる。しかし、複雑な意味規則の定義はユーザにとって困難な作業であり、それに基づく網羅性判定のデータの収集もコスト高になるであろう。従って、簡単な意味規則に基づく網羅性判定法と構文的網羅性判定法をいかに適切に組合せて使用したらよいかを検討する必要がある。

参考文献

[1] Setzer, V.W., Non-recursive Top-down Syntax Analysis, Software-P&E, vol.9, 1979, pp. 237-245.

[2] Wirth, N., Algorithms + Data Structures = Programs, Prentice-Hall, 1976.

[3] 上原他, コンパイラのテストプログラム網羅性判定ツール, 情報処理学会第22回全国大会.

[4] 上原他, コンパイラのテストプログラム網羅性判定について, 情報処理学会第24回全国大会.

付録　生成規則の複雑度

非終端記号 A の生成規則 p_A に対して、k 階の複雑度 $C^k_{p_A}$ を次のように定義する。

(i) 生成規則 p_A : $A ::= t_1 | t_2 | \cdots | t_\ell$ 。,
t_i は項 $(i=1 \ldots \ell)$ に対して、

$$C^k_{p_A} = \sum_{i=1}^{\ell} C^k_{t_i} \qquad (k = 0,1,\ldots)$$

(ii) 項 t : $f_1 f_2 \cdots f_m$, f_i は因子 $(i=1 \ldots m)$ に対して、

$$C^k_t = \prod_{i=1}^{m} C^k_{f_i} \qquad (k = 0,1,\ldots)$$

(iii) 因子 f に対して、

f が終端記号の場合

$$C^k_f = 1 \qquad (k = 0,1,\ldots)$$

f が非終端記号 B の場合

$$C^k_f = \begin{cases} 1 & (k=0) \\ C^{k-1}_{p_B} & (k = 1,2,\ldots) \end{cases}$$

f が繰返し部 $(\alpha_1 | \alpha_2 | \cdots | \alpha_m)+$ の場合

$$C^k_f = X + X^2 \qquad (k = 0,1,\ldots)$$

f が繰返し部 $(\alpha_1 | \alpha_2 | \cdots | \alpha_m)*$ の場合

$$C^k_f = 1 + X + X^2 \qquad (k = 0,1,\ldots)$$

但し、α_i は項 $(i=1 \ldots m)$、$X = \sum_{i=1}^{m} C^k_{\alpha_i}$ とする。X は繰返し部の1回の繰返しの複雑度を表わし、X^2 は2回の繰返しの複雑度を表わしている。またfが（式）* のときは0回の繰返しも考慮されている。