C言語プログラム検査システムの設計

5T−6

宮武　圭子　　瀧塚　孝志

国際電信電話株式会社研究所

1. はじめに

プログラム開発においては、保守や移植を考えた開発が重要である。同じプログラムであっても、計算機やコンパイラが異なるとエラーが出たり出なかったりする。また、プログラムスタイルが統一されていないと、保守や移植が困難になる。更に、見通しの悪いデータ構造や、モジュールの複雑な呼び合いも保守を困難にする。このような問題を起こす部分を検出するために、表記検査ツール、モジュール評価ツールの2つのツールより成るC言語プログラム検査システムを現在開発している。本稿では、このシステムの概要について述べる。

2. 設計方針

現在、通信プログラムの多くは、システム記述に適したC言語を用いて開発されている。C言語のANSI標準規格が定められてから、ANSI準拠のC言語コンパイラが多くの計算機で利用可能となりつつある。そこで、本システムの対象言語としてANSI Cを選択した。

本ツールは、独立したツールとして、OSに依存しない形式で開発する。しかし、社内のソフトウェア開発では、しばしばVMSが使用されるので、VMSのC言語で書かれたプログラムにも適用出来るようにする。将来は、オブジェクト指向記述も可能なC++等の言語に拡張する予定でいる。

ツールの機能評価のための試作版は、ソースプログラムが利用出来るGNUのCコンパイラを改修して作成する。追加、修正すべき機能が明らかになった後、改修では実現が困難な機能も含めて、専用のツールを開発する。

2.1 表記検査ツール

UNIXには、lintというツールがあり、UNIX上でプログラムを開発する際によく使われる。lintは、関数の引数や、戻り値の使用、未定義値の使用、未使用変数、意味のない文等の検査を行う。

lintには、移植性の検査のための様々なヒューリスティックが取り入れられている。しかし、lintで警告が出ないような表記に対しても警告を出すコンパイラがあるため、検査機能を強化する必要がある。

lintは、K&RのC言語で書かれたプログラムを対象としているので、ANSI Cで書かれたプログラムを検査することが出来ない。そこで、lintをANSI Cに適合するように拡張する必要がある。

A Design of Evaluation System for C Programs
Keiko MIYATAKE, Takashi TAKIZUKA
KDD R&D Laboratories

プログラムの誤りを少なくするためには、Adaのような強く型付けされた(strongly typed)言語が有効である。ANSI Cにおいても、typedefで導出された型に対し、同様の強い検査機能を提供することにした。

2.2 モジュール評価ツール

プログラムの複雑性を定量的に評価する手法が研究されている。複雑性には、モジュール内を見て評価する内面的複雑性と、タスク(プロセス)の相互作用を評価する外面的複雑性がある[1]。内面的複雑性の尺度の主なものとして、McCabeのComplexity Measure[2]や、HalsteadのProgramming Effort[3]がある。

Complexity Measureはプログラムを制御構造に基づいてグラフ化し、それを解析し、評価する。しかし、C言語のswitch文のような多分岐を過剰に評価することが知られている[4]。

Programming Effortは、プログラム中で使われる演算子と被演算子の数と種類によって複雑性を評価する。C言語では++a, a++, a+=1, a=a+1のように、同じ動作を複数の種類の演算子を用いて書くことができる。これらを同じ種類の演算子と見なす必要がある。

C言語では、#defineで定義されたマクロがプリプロセッサによってマクロ展開される。プログラムを書いたり読んだりするときには、マクロも通常の関数や定数と同一の複雑性である。しかし、展開後のプログラムを評価すると、複雑性が増大する問題点がある。

3. C言語プログラム検査システム

3.1 表記検査ツール

表記検査ツールには、lintの機能を基本として、さらに次のような機能を追加する。

(1)各種コンパイラにある検査:「if(x=y)」を「if(x==y)」の間違いではないかとするような落し穴の検査。データフロー解析を用いた未定義値や未使用変数の検査。「a=+1」を「a=(+1)」とは解釈せず、「a+=1」と解釈するような古い形式に対する検査。

(2)ANSIの変更点:ANSI Cでは、それ以前のCとスコープ・ルールが異なっている。ある関数内でint型でない外部関数を宣言しても、他の関数からは宣言が見えなくなるため、ANSI Cでコンパイルすると動かなくなるプログラムがある。

また、ANSI Cでは、関数のプロトタイプ宣言があるため、未定義外部関数に警告するコンパイラがある。

したがって、プロトタイプ宣言することを前提とし、未定義関数の使用には警告を出すことにする。プロトタイプ宣言を前提とすることにより、外部関数の引数や型を検査するために利用していたlintライブラリが不要に

なる。
(3) プログラム記述形式：同じ無限ループの表記にも、for(;;) や do;while(1) や while(1) のように異なった表記の仕方がある。このような表記を、少なくとも同じファイル内で統一するような、プログラム記述形式の検査を行う。
(4) 強い型検査：typedef を用いて導出される型についての強い型検査を行う。例えば、

```
typedef int a;
typedef int b;
```

としたときにａとｂを異なる型とみなす。

なお、エラーを起こし易い記述のうち、回避が困難な記述や、構文／意味解析で検出が可能な記述は、プログラミング手引書を作成することにより解決する。

3.2 モジュール評価ツール

Ｃのマクロでは、

```
#define then {
#define fi }
```

のように記述することが出来るので、マクロを展開せずに、構文解析を行うことは不可能である。

マクロの中が文、または文の並びとして解析出来ない場合は、展開する必要がある。マクロの最後の文が if 文であるとき、展開しないと後の else を解析出来ない。マクロの最後の if 文は、「if(マクロ名)」のように展開し、後ろに else が来ても解析出来るようにする必要がある。

マクロを展開しないで評価を行うには、プリプロセッサとパーサの変更点が大きいため、試作版では関数とみなして評価を行うことにした。

マクロは、しばしばインライン展開をすることによって、関数として実装するよりも、実行速度を高めるために用いられる。C++には inline というキーワードがあり、C++を利用するようになれば、関数的なマクロの多くが inline 関数で書かれることが期待される。

4. 実装状況

フリーソフトウェアの GNU の ANSI C コンパイラとプリプロセッサを改良して表記検査ツールを作成している。VMS の C 言語である VMS/C は ANSI C を拡張したものであり、拡張部分に対する変更を完了した。

4.1 プリプロセッサ

「#pragma standard」と「#pragma nostandard」による処理系に依存した前処理を行うように改良した。「#pragma standard」で指定されている領域では、ANSI C 以外の記述を行うと、警告を行う。

「#include token」と書いた場合、ANSI C では token を展開後、インクルードする。VMS では、この token を、テキストライブラリのキーとして用いている。＜token.h＞と解釈しても、エラーメッセージが変わったり、コンパイル速度が遅くなる位で問題がない。「#pragma standard」モードで＜token.h＞と解釈しなければならないときは、警告を出す。

VAX,VMS 等の VMS/C 特有の定数を定義した。試作版では、データディクショナリは扱わない。

4.2 表記検査ツール

GNU には、VMS 版の C コンパイラがあり、VMS/C 特有の予約語の解析を避けるため、GNU のための定義ファイルが用意されている。しかし、既存の通信プログラム自体が VMS/C 特有の予約語を含んでいるので、GNU ではコンパイル出来ない。

そして、VMS/C 特有の予約語である、_align 等の storage class modifier や、variant_struct 等の variant keywords、globalvalue 等の global storage class modifier に対し、解析を行えるように GNU の C コンパイラを改良した。

また、「a=+1」のような古い形式の C 言語の記述に対しては、警告を出せるようにオペレータを追加した。

4.3 定義ファイルの作成

VMS には多くのライブラリが用意されている。このうちシステムライブラリには 3000 個以上の関数がある。実際の通信システムの記述に使われていた 130 個の関数に対し、その定義ファイルを作成した。その結果、よく使われるシステムコールや、VMS 特有のファイルアクセス関数について、ライブラリ関数の定義が出来た。

4.4 モジュール評価

モジュール評価の機能については、データフロー解析部に McCabe の Complexity Measure の評価プログラムを組み込んだ。現在、疑似コード生成部 (RTL) に Programming Effort を評価するプログラムを組み込んでいる。

5. おわりに

C 言語プログラム検査システムの機能について説明した。表記上の検査ツールについては、今年度中に lint と同程度の検査機能を実現し、その後機能を追加する。

モジュール評価ツールの評価結果を用いて、エラーの発生度や保守性を予測したり、設計の善し悪しを評価することが出来る。しかし、それだけでは利用度が低い。複雑性を低下させるモジュール分割やデータ構造の構築を提示する手法を開発後、マクロ展開をしない版を作成する予定でいる。

最後に日頃御指導頂く KDD 研究所小野所長、浦野次長、小西主幹研究員に感謝する。

参考文献

[1] “Software Complexity and Its Impact on Sofrware Reliability”, Ken S. Lew, Tharam S. Dillon, Kevin E. Forward, IEEE Trans. SE Vol.14 No.11 Nov.1988.
[2] “A Complexity Measure”, Thomas J. McCabe, IEEE Transactions on Software Engineering, SE-2(4), Dec 1976.
[3] “Elements of Software Science”, Maurice H. Halstead, The Computer Science Library, 1988.
[4] “プログラムの複雑性評価”, 平野行芳, 大場充, 谷津行穂 (日本 IBM), 情報処理学会論文集, ソフトウェア工学 26-4(1982/10/26)