社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2007－ＳＬＤＭ－128（11）
2007／1／18

Cコンパイラ用テストスイートおよびその生成ツールtestgen

内山　裕貴†　　引地　信之††　　石浦菜岐佐†　　永松　祐二†††

† 関西学院大学 理工学部　〒669-1337 兵庫県三田市学園 2-1
†† SRA　〒171-8513 東京都豊島区南池袋 2-32-8
††† 関西学院大学 (現在 日本総合研究所)
E-mail: †{uchiyama-y,ishiura}@ksc.kwansei.ac.jp, ††hikichi@sra.co.jp

あらまし　コンパイラのテストは，一般にテストスイート (コンパイラの入力となるプログラムの集合) を用いて行う．コンパイラ用テストスイートには類似したコードが多く含まれる傾向があり，プログラムの保守性という点からは必ずしも望ましくない．また，テストに用いるコンパイラやシミュレータの変更やテスト項目の指定などに多くの手作業が必要になることがある．本稿ではこれらの問題を解決する一手法としてテストプログラムの元となる「テンプレートファイル」と設定ファイルからテストスイートを生成する手法を提案する．本手法を既存のテストスイートに適用した結果，テストプログラムの記述量を約 60%に削減することができた．

キーワード　C 言語, コンパイラ, テストスイート, gcc

Test Suite of C Compilers and Its Generating Tool "testgen"

Yuki UCHIYAMA†, Nobuyuki HIKICHI††, Nagisa ISHIURA†, and Yuji NAGAMATSU†††

† Kwansei Gakuin University,　Gakuen 2-1, Sanda, Hyogo, 669-1337 Japan
†† SRA,　Minami Ikebukuro 2-32-8, Tosima-ku, Tokyo, 171-8513 Japan
††† Kwansei Gakuin University (now with the Japan Research Institute)
E-mail: †{uchiyama-y,ishiura}@ksc.kwansei.ac.jp, ††hikichi@sra.co.jp

Abstract　The testing of compilers is usually done using a test suite, or a collection of programs fed into the compilers. The test suite for compilers often includes many similar codes, which is not desiarable from the viewpoint of the program maintainability. It sometimes takes manual editing of many files to switch compilers and simulators or to select specific test sets. As an approach to solve those problems, we propose in this article a method of generating test suites from "template files," which are reduced description of the test programs, and a configaration file. We demonstrate that the total number of the lines for describing an exisiting test suite is reduced to 60% by the proposed method.

Key words　C language, compiler, test suite, gcc

1. はじめに

組み込みシステムに課せられる性能，電力，コスト等の厳しい制約を満たす一つのアプローチとして，近年 ASIP (Application Specific Instruction Processor) 等ターゲットに特化したプロセッサを設計する機会が増えている．これに伴い，新しいプロセッサに対するコンパイラを開発する機会も増えており，そのテストが重要な課題となる．

コンパイラのテストは，入力となるプログラムの集合を準備し，そのプログラムのコンパイルがエラーなく行えるか，コンパイルされたプログラムが期待通りの動作をするかを確認することにより行う．コンパイラの入力となるプログラムをテストプログラム，その集合をテストスイートと呼ぶ．

コンパイラのテストスイートには，類似したプログラムが多く含まれる傾向がある．例えば，変数への代入処理が正しく行えるかをテストするプログラムでは，同様の処理を異なる型や異なる記憶クラスの変数に適用するため，変数宣言部分のみが異なる類似したテストコードが多数存在することになる．このように重複の多いプログラムは，変更を行うときの影響が複数箇所に及ぶため，保守性の観点からは望ましくない．また，テストを行うコンパイラやシミュレータの変更，テストスクリプトの動作の変更，テストスイートの一部のみを実行する指定などを行うために，幾つものファイルを変更しなければならないケースが生じる．

本稿では，これらの問題を解決するための一手法として，テストプログラムの元となる「テンプレートファイル」と設定ファイルからテストスイートと必要なテストスクリプトを生成する手法を提案する．この手法により，テストプログラム記述量の減少による保守性の向上，設定ファイルによる環境カスタマイズの容易化，部分的なテストスイートの生成を実現する．

本手法を SRA で開発された C コンパイラ用テストスイート (以下 SRA テストスイート) に適用し結果，テストプログラムの記述量を約 60%に削減でき，異なった環境で使用する際の設定変更が容易に行えるようになった．

2. C コンパイラ用テストスイート

C コンパイラのテストスイートは，一般に C 言語で書かれたテストプログラムの集合から成るが，テスト実行や結果の解析を行うスクリプトを含めてテストスイートと呼ぶこともある．テストプログラムの数はテストスイートによって異なるが，数千から数万を超えるものまであり，非常に大規模になる．

2.1 既存のテストスイート

(1) GCC テストスイート (注1)

GCC (GNU Compiler Collection) に付属のテストスイートは，GCC のテストを主な目的として作成されたテストスイートである．テストプログラムのファイル数は約 2,500 個で，総行数は 23,000 行である．テストのフレームワークに dejagnu (注2) を用いている．

テストプログラムは C 言語の仕様として標準で定められている言語仕様のテストに加え，GCC 独自の言語拡張のテストも含む．そのため，GCC 以外のコンパイラに適用しようとすると，多くの修正が必要となる．

また，テスト結果の出力に出力関数を用いており，ライブラリの存在を前提としている．しかしコンパイラ開発時にはライブラリが利用できない場合が多く (ライブラリが完全にコンパイルできない等の理由による)，その場合には何らかの出力機能を別途作成する必要がある．

(2) quest (注3)

quest は，関数呼び出しの引数渡しのテスト用の C プログラムを自動生成するツールである．

quest を実行すると引数の型と個数がランダムな関数，およびその関数を呼び出す関数が記述されたテストプログラムが生成される．生成された関数の引数は基本型，ポインタ，配列，構造体，およびこれらの複雑な組み合わせを含む．quest はテスト項目を関数呼び出し時の引数渡しに絞ってコンパイラの完成度を高めることに適しているが，その他の構文はほとんどカバーしない．

(3) SRA テストスイート

SRA テストスイートは多くのコンパイラへの適用，およびコンパイラの開発段階での適用を目的とした，汎用性の高いテストスイートである．テストスイートのファイル数は約 9,300 個，テストプログラムの総行数は約 48 万行である．

(注1)：http://gcc.gnu.org/

(注2)：http://www.gnu.org/software/dejagnu/

(注3)：http://www.st.cs.uni-sb.de/ lindig/src/quest/

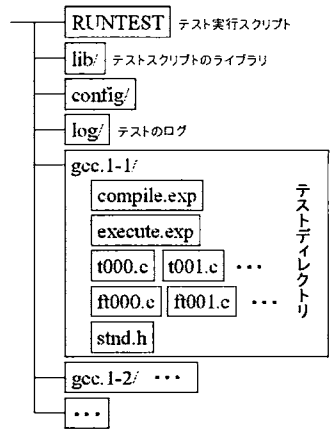

図 1 SRA テストスイートのディレクトリ構成

GCC テストスイートが対象となるコンパイラを限定しているのに対し，SRA テストスイートは C 言語の標準的な機能のテストに重点を置いている．

テストスイートは基本的な機能のテストを行う小さなプログラムを多数含んでおり，コンパイラ開発時でのテストでエラー原因を特定しやすい．また，テストプログラムはファイル名で浮動小数点型，整数型の区別をできるようになっており，問題の特定や一部の型のみのテストが行いやすい．

テスト結果の出力はマクロ定義されており，ライブラリが利用できない場合には，例えばメモリの特定番地へのデータの書き込み等に変更できる．そのため，コンパイラ開発の初期段階でライブラリが未完成であっても利用が可能である．

図??に SRA テストスイートのディレクトリ構成を示す．SRA テストスイートは C 言語のテストプログラム集合，テスト実行スクリプト，ライブラリ等から構成される．フレームワークには GCC テストスイートと同様に dejagnu を用いており，dejagnu のコマンドとオプションが記述された “RUNTEST” スクリプトを実行することによりテストを行う．RUNTEST は各ディレクトリに配置された compile.exp (コンパイルのテストスクリプト), execute.exp (実行のテストスクリプト) を次々に呼び出すことによりコンパイラのテストを行う．

2.2 SRA テストスイートの問題点

SRA テストスイートのテストプログラムには，差異の少ないテストプログラムが数多く含まれている．図??にその一例を示す．これは変数への代入が正しく行われているかをテストするプログラムであるが，(1) と (2) を比較すると変数宣言が異なっているだけで，それ以外の部分は共通である．もしテストプログラムに対する変更が必要になった場合には，同様の変更を多数の箇所に対して行わなければならない．

また，テスト環境のカスタマイズを行うには多くのファイルを修正する必要が生じる．例えば，クロス開発環境で，実行にシミュレータを用いる場合，各ディレクトリに存在する実行ス

```
#ifdef SYSDEP_H
#include "sysdep.h"
#endif
#include "stnd.h"
main(){
  int Var;
  itest = NO;
  Var = OK;
  itest = Var;
  if(itest==OK)
    printok();
  else
    printno();
  return (0);
}
```

(1) gcc.1-1/t000.c

```
#ifdef SYSDEP_H
#include "sysdep.h"
#endif
#include "stnd.h"
main(){
  char Var;
  itest = NO;
  Var = OK;
  ictest = Var;
  if(itest==OK)
    printok();
  else
    printno();
  return (0);
}
```

(2) gcc.1-1/t012.c

図 2　共通部分の多いテストプログラムの例

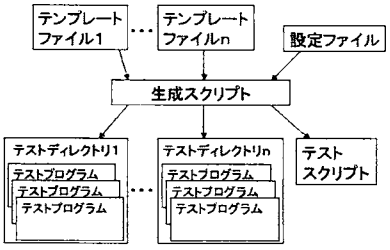

図 3　テストスイート生成の流れ

クリプトである execute.exp をすべて修正しなければならない.

3. 提案するテストスイート生成の手法

本稿では 2.2 節で述べた問題点を解決する一手法として，テストプログラムの元となる「テンプレートファイル」と設定ファイルからテストスイートを生成する手法を提案する．この手法は，

- テストプログラム記述量の減少による保守性の向上
- 設定ファイルによる環境カスタマイズの容易化
- 部分的なテストスイートの生成

を目的とする．

図??に提案するテストスイート自動生成の流れを示す．テスト生成ツールはテンプレートファイルと設定ファイルを入力とし，テストプログラム集合，およびテストスクリプトを生成する．テンプレートファイルは，一つのテストディレクトリに存在する全てのテストプログラムやスクリプトファイルを一つにまとめて記述したファイルである．設定ファイルには環境情報を記述し，その変更により生成されるテストプログラムや実行スクリプトを容易に変更できるようにする．

3.1 テンプレートファイル

テンプレートファイルは，複数のテストプログラム中の重複部分のくくり出しと繰り返し構文の適用により，記述量を削減してテストスイートの保守性の向上を図るものである．

図??にテンプレートファイルの例を示す．1〜3 行目はコメントであり，生成されるテストプログラムには影響しない．5〜10 行は変数の定義である．ここでは HEAD という変数を定義

```
 1 @comment
 2 変数の代入のテスト
 3 @end_comment
 4
 5 @multiline HEADER
 6 #ifdef SYSDEP_H
 7 #include "sysdep.h"
 8 #endif
 9 #include "stnd.h"
10 @end_multiline HEADER
11
12 @for VARIABLE 'int' 'short' 'char'
13 @for TYPE '' 'static' 'register'
14 @file t???_$MODIFIER_$VARIABLE.c
15
16 $HEADER
17 main(){
18   $MODIFIER $TYPE Variable;
19   itest = NO;
20   Variable = OK;
21   itest = Variable;
22   if(itest == OK) printok();
23   else            printno();
24 }
25
26 @end_file
27 @end_for VARIABLE
28 @end_for MODIFIER
29
30 @run compile.exp execute.exp
```

図 4　テンプレートファイルの例

しており，15 行目の $HEAD は 5〜7 行目の内容で置換される．12, 13〜27, 28 行目の@for から@end_for は，その間に記述されたコードを，変数の置換を行いながら繰り返し生成する．@for はネストが可能であり，この例では 9 通りの異なったプログラムが生成される．14〜26 行目の @file から @end_file で囲まれた部分が一つのテストプログラムに対応する．@file の後にはファイル名を記述し，"??" はテストプログラムの通し番号で置換される．

図??はファイル名が t001_static_int.c で生成されたテストプログラムである．プログラムは図??の 16 行目〜24 行目に相当し，13 行目の$HEADER，15 行目の$MODIFIER と$TYPE がそれぞれ置換されている．

3.2 設定ファイル

図 6 に native コンパイラ (gcc) をテストするめの設定ファイルの例を示す．1 行目から 3 行目は順に，テストを行うコンパイラ，ターゲット，実行時のシミュレータ (空白の場合は生成

```
#ifdef SYSDEP_H
#include "sysdep.h"
#endif
#include "stnd.h"
main(){
  static int Variable;
  itest = NO;
  Variable = OK;
  itest = Variable;
  refer(itest);
}
refer(int itest){
  if(itest == OK) printok();
  else            printno();
}
```

図 5 生成されるテストプログラム

```
 1 CC:gcc
 2 TARGET:gcc
 3 SIMULATOR_NAME:
 4 TOOL:gcc
 5 LIBS:
 6 LDFLAGS:
 7 FILSET:FILSET
 8 OBJ:.
 9 LOG:log
10 TESTDIR:testsuite
```

図 6 設定ファイルの記述例 (native 環境)

```
 1 CC:m32r-unknown-elf-gcc
 2 TARGET:m32r-gcc
 3 SIMULATOR_NAME:m32r-unknown-elf-run
 4 TOOL:gcc
 5 LIBS:
 6 LDFLAGS:
 7 FILSET:FILSET
 8 OBJ:.
 9 LOG:log
10 TESTDIR:m32r-testsuite
```

図 7 設定ファイルの記述例 (m32r クロス環境)

されたコードが実行可能ファイルとして実行される) を指定している. 4 行目はテストを行うディレクトリの名前を指定しており, ここに定義された文字列で始まるディレクトリをテスト対象とする. 5 行目から 7 行目ではテスト実行時のオプションやライブラリ等の指定を行う. 8 行目から 10 行目ではテストスイート, ログ等を出力するディレクトリを指定する.

図 7 はクロスコンパイラをテストするための設定ファイルの例である. この設定では, コンパイラに m32r-unknown-elf-gcc, ターゲットに m32r-gcc, シミュレータに m32r-unknown-elf-run を指定している.

一つの設定ファイルに必要な情報をすべて記述することにより, テストスイートのカスタマイズ性を向上できる.

3.3 部分的なテストスイートの生成

テストスイート生成時に, 入力となるテンプレートファイルを選択することにより, 部分的なテストスイート生成を可能とする. また, 生成時のオプションとして整数型のみを含むテスト, 浮動小数点型のみを含むテストを-f, -i オプションにより指定できるようにする.

4. testgen の実装と利用例

以上の提案手法に基づくテスト生成スクリプト "testgen" を Perl 5 で実装した. testgen は Linux Fedora Core 4, Cygwin 1.5, Max OS X, Debian GNU Linux 上の Perl 5.8.7 以上で動作する.

次に, testgen を SRA テストスイートに適用した. SRA テストスイートと同じテストスイートを生成するテンプレートファイル, および設定ファイルを作成した. テンプレートファイルから正しくテストスイートが生成されることは, 生成された全テストプログラムを "diff" コマンドにより比較し, テキスト的に異なるものについては目視での確認をすることにより確かめた.

表??は SRA テストスイートのテストプログラムの行数と, 同じテストプログラムを生成するテンプレート記述の行数を比較たものである. 全体としては行数が約 60%に減少した.

表 1 記述量の比較

ディレクトリ名	元のテストスイート (行)	テンプレート記述 (行)
gcc.1-1	1,822	965
gcc.2-1-01	3,351	999
gcc.3-1	6,080	5,623
gcc.4-1	2,364	2,149
gcc.5-1	452	310
gcc.6-1-01	2,152	2,259
全体	485,232	284,619

SRA テストスイートは, ファイル名のマッチングで整数型のテストプログラムのみを抽出できるようになっているが, 一部の整数型のテストプログラムのヘッダ中に浮動小数点型変数の宣言が含まれるという誤りが発見された. この問題の修正をテストスイートに対して直接行うには約 3,000 個のプログラムファイルの修正が必要となるが, テンプレートファイルに対して修正を行えば 100 ファイルで済んだ. このことからも, テストスイート生成の枠組みは, テストスイートの保守性の向上に有用であると考える.

生成されたテストスイートにより，コンパイラのテストが正常に行えることを下記の環境で確認した．

- Mac OS X 10.3.9/GCC 3.3.4
- Cygwin on Win2k/GCC 3.3.4
- Fedora Core 4/GCC 4.0.0
- Debian GNU Linux/GCC 3.3.5

生成されたテストを全て実行すると native gcc (3.4.6), Fedora Core 4 Linux (Pentium4 2GHz, 512MB Mem) 環境で1時間弱の時間を要する．また，native gcc (3.4.4), Cygwin (1.5)/WindowsXP SP2 (PentiumM 1.5Ghz, 1024MB Mem) 環境は3時間弱が必要となる．

5. まとめ

本研究ではテンプレート記述からテストスイートを自動生成する手法の提案と，その SRA テストスイートへの適用について述べた．

testgen (テストスイートを含む) は

http://ist.ksc.kwansei.ac.jp/~ishiura/testgen/

で公開している．

このテストスイートを改良すべき点としては下記が挙げられる．テストスイート生成によりこれらの点が解決できないか，今後検討を進める予定である．

(1) ほとんどのテストプログラムで使っている値は 0, 1 だけなので，境界値を含む多くの値によるテストの追加が望まれる．境界値はコンパイラやターゲットプロセッサ等によって異なるので，設定ファイルとテストスイート生成の組み合わせが有効と考える．

(2) このテストスイートを全て実行した場合に要する時間は開発時に何度もテストを行うことを考えると大幅に短縮することが望まれる．複数の機能のテストを一つのプログラムにまとめることができればテストの時間が短縮できるので，テストスイート生成によるその実現を検討したい．

(3) 現在のテストプログラムはすべて K&R C で書かれているが，ANSI C や C99 に対応する必要がある．この際にもテストスイート生成での解決が有効と考える．

謝　辞

本研究は，IPA 2004 年度オープンソースソフトウェア活用基盤整備事業のプロジェクト「C コンパイラ向けテストスイート生成ツールの開発」に採択されたものである．

本研究に際し，多くの助言や協力を頂いた関西学院大学の森本剛徳氏，粟津裕亘氏をはじめとする石浦研究室の諸氏に感謝します．

文　献

[1] Don Libes: "Exploring Expect", O'REILLY, (1995).
[2] B.W. カーニハン/D.M. リッチー著 石田晴久訳: "プログラミング言語 C 第2版 ANSI 規格準拠", 共立出版株式会社, (1989).