呼出し規約テスター

（この資料は、SuperTest Vermeer Update 4 の Solid Sands Test Platform Manual の 10 章からの抜粋です）

コンパイラの呼出し規約は、関数が、それを呼び出した関数からどのようにしてパラメータを受け取って、その結果を呼び出した関数にどのように返すか、を規定するものです。呼出し規約は、言語が規定している抽象的な概念を実装したものですので、言語規格では定義されていません。関数を呼び出す動作をできるだけ効率的にすることを目的としており、この実装は非常に複雑なものになることがあります。このことは、多くの場合に呼出し規約がコンパイラのレジスタ割当てと密接に関連していることを意味します。

それゆえコンパイラの検証では、呼出し規約が正しく実装されているかどうかをテストすることが重要です。このために `calltest` コマンドとして用意された呼出し規約テスター[1] が使用できます。呼出し規約テスターは、コンパイル・リンクできる C 言語のテストファイル群を生成して実行時し、正しさをセルフチェックします。

呼出し規約テスターは `strun` テストドライバと統合されており、以下のコマンドで起動します（オプションの `-c` が違いです）：

```
strun -c default.cfg
```

これにより、コンフィグレーションファイル `target/calltest.cfg` で定義されているパラメータで呼出し規約テスターが実行されます。コンフィギュレーションファイルは変更することができ、これについては後で詳しく説明します。

呼出し規約テスターは、C 言語のソースファイルのペアを生成します。このペアの一方（呼び元）にはランダムに生成された値を実パラメータに設定した関数の呼出しがあり、もう一方（呼び先）には呼び出される関数の実装があって、入力となる仮パラメータが予期される値であるかどうかをチェックします。さらに、その関数はランダムに生成した値を返し、その戻り値が呼び元のファイルにある呼出し元でチェックされます。

生成される実引数リストと戻り値は、ポインタ、配列、可変引数、共用体や構造体（ネストした型やビットフィールドも含んで）が混在した（呼出し規約の観点で）複雑な型で構成されます。

呼出し規約テスターの実行ではオプションにより、ペアの一方のファイルをあるコンパイラで、もう一方のファイルを別のコンパイラ（または同じコンパイラの別のバージョン）でコンパイルするようコンフィグレーションできます。異なるコンパイラでコンパイルされたコンポーネントからなるプログラムが正しくリンクされて実行でき、ライブラリが使用できることを保証するためには、双方のコンパイラが同じ呼出し規約を実装していることが必要です。この

[1] SuperTest の MISRA 専用のコンフィグレーションには呼出し規約テスターは含まれません。

設定により、一緒にリンクされて実行されるときに双方のコンパイラが同じ呼出し規約を使用していることが検証されます。詳しくは 10.2.1 項を参照ください。

生成された C 言語ソースファイルのペアの例を以下に示します。ここで、表 10.1 は呼出し元で、表 10.2 は呼出し先を示します。ここでは 2 つの値 `-47` と `-102` が、呼出し元によって型 `t0` の文字配列 `v0` に書き込まれ、関数 `func` に実パラメータとして渡されます。呼出し先(`func`)では、それぞれ同じ値が得られているかがチェックされます。これは、戻り値のない簡単な例です。

```
#include "def.h"
typedef union{signed char v0[2];}t0;
extern void func(t0 v0);

MAIN {
        t0 v0;
        v0.v0[0] = -47;
        v0.v0[1] = -102;
        func(v0);
}
```

表 10.1 呼出し元

```
#include "def.h"
typedef union{signed char v0[2];}t0;


void func(t0 v0) {
        CVAL_VERIFY(v0.v0[0] == -47);
        CVAL_VERIFY(v0.v0[1] == -102);
}
```

表 10.2 呼出し先

呼出し規約テスターは生成する実パラメータリストや使用される型の種類について高度にコンフィグレーションできます。これについて 10.1 節で詳しく説明します。10.2 節で `strun` テストドライバによる呼出し規約テスターの使用について、10.3 節でランダムジェネレータのコンフィグレーションについて説明します。

10.1 コンフィグレーションファイル

ジェネレータが使用する構成設定ファイルは行ベースのもので、設定オプションは複数行にまたがることができません。「`#`」でコメントが開始され、その行末（改行文字は含みません）まで続きます。1024 文字を超える行は切り縮められます。コンフィグレーションファイルの各行は次の形式です：

```
<option> = <value>
```

ここで、`<option>`はコンフィグレーションのオプション、`<value>`はそれに対応する値です。

10.1.1 コンフィグレーション設定値の型

コンフィグレーション設定値の型はオプションに依存します。10.1.2 項でコンフィグレーションのオプションを説明する前に、それらの型を説明します。

数

```
[-][0-9]+
```

数は十進数字の並びで先頭にマイナス記号がつくことがあります。

型マスク

([+-]*<type-id>)*

型マスクは、生成プロセスで使用される特定の C の型のリストを指定するものです。表 10.3a ですべての使用可能なプリミティブ型を示し、10.3b で使用可能な集成型を示します。型マスクは、空白、もしくは符号「+」か「-」で区切られます。リストは左から右に評価され、符号が続く場合は最後のものが採用されます。「+」は現在の型マスクに 1 つ型を加え、「-」は現在の型マスクから 1 つ型を減じます。たとえば、利用可能な集成型に all がありますが、-all は現在の型マスクを空にし、+all は型リストがすべての型を含むようにします。

型マスクは常に-all もしくは+all で始まり、それぞれベース集合として何もなしか、すべてかを使用するようにします。たとえば、float 型か float 型を含む struct 型だけを生成するには次の型マスクを使用します。

```
-all+struct+float
```

同様に、union 型と long 型以外のすべての型を使用するテストを生成するには次の型マスクを使用します。

```
+all-union-long2
```

型 id	対応する C の型	実装
schar	signed char	8 ビット
sshort	signed short	16 ビット
sint	signed int	16/32 ビット
slong	signed long	32/64 ビット
slong2	signed long	64 ビット
uchar	unsigned char	8 ビット
ushort	unsigned short	16 ビット
uint	unsigned int	16/32 ビット
ulong	unsigned long	32/64 ビット
ulong2	unsigned long long	64 ビット
bool	_Bool	1 ビット
float	float	最小限 10^19
double	double	最小限 10^19
ldouble	long double	最小限 10^19
dpointer	void *	ulong と同じサイズ
fpointer	void (*)(void)	
struct	struct ..	
union	union ..	
array	.. [N]	

(a) プリミティブ型

型 id	集成型
all	全てのプリミティブ型
signed	schar, sshort, sint, slong slong2
unsigned	uchar, ushort, uint, ulong, ulong2
integral	signed, unsigned, bool
char	schar, uchar
short	sshort, ushort
int	sint, uint
long	slong, ulong
long2	slong2, ulong2
floating	float, double, ldouble
pointer	dpointer, fpointer
composite	record, union, array

(b) 集成型

表 10.3　型マスクで使用できる型

10.1.2 コンフィグレーションオプション

次のコンフィグレーションオプションをサポートしています。以下では、構成設定オプションの名前をリストし、オプションの値は( )内に示します。[ ]内がデフォルトのオプション値で、[ ]の後ろの( )内でオプション値に対する追加の制約が指定されます。

max_test (number) [300] (max_test > 0)

生成するテスト数の最大値

max_param (number) [8] (max_param > 0)

生成される関数の仮パラメータ個数の最大値

new_mask (type mask) [+all]

仮パラメータと戻り値に使用できる型。この型には最低限 1 つのプリミティブ型を含むことが必要です。C90 には long と _Bool 型はないので、C90 のコンパイラのテストではこのオプションを除くよう修正が必要です

data_model (number) [32] (data_model == 16/32/64)

整数型とポインタ型の値を生成する場合に使用するデータモデルで、16 ビットモデルでは 16 ビットの int、32 ビットの long、64 ビットの long long です。ポインタは long と同じサイズです。32 ビットモデルは ILP32(32,32,64)に対応し、64 ビットモデルは LP64(32,64,64)です

min_array (number) [2] (min_array > 0)

配列の長さの最小値

max_array (number) [17] (max_array >= min_array)

配列の長さの最大値

mask_bitfield (type mask) [-all+int+bool] (must only contain integral types)

ビットフィールドに対して使用できる型で、すべて汎整数型です。型が指定されない場合、ビットフィールドは生成されません。デフォルトは C99（以降）準拠です。C90 には _Bool がありませんので、C90 準拠のコンパイラをテストする場合にはこのオプションを変更しなければなりません。このオプション値は new_mask オプションで定義されない型を含んではなりません

min_vararg (number) [0] (min_vararg >= 0)

関数プロトタイプの可変長引数リストの仮パラメータの個数の最小値

max_vararg (number) [5] (max_vararg >= min_vararg && max_vararg < max_param)

関数プロトタイプの可変長引数リストの仮パラメータの個数の最大値。min_vararg と max_vararg 両者が 0 に設定された場合、可変長引数関数は生成されません

min_bitfield (number) [1] (min_bitfield > 0)

生成されたビットフィールドのビット数の最小値

max_bitfield (number) [31] (max_bitfield >= min_bitfield)

生成されたビットフィールドのビット数の最大値

random_seed (string) [""] (four integers seperated by spaces)

テスト生成の開始時に使用するランダムジェネレータのシードで、このコンフィグレーションを使用した各実行が同じテストを生成するようにします。デフォルトでは、この値は空です。つまり、固定されたシードは使われないということです。ランダムジェネレータのシードを設定する方法については、10.3 節を参照してください

10.2 呼出し規約テスターの使用

呼出し規約テスターは strun テストドライバを使用し、--calltest（もしくは -c）オプションをつけてコマンドラインから呼び出します。

```
strun --calltest default.cfg
```

加えて、トップレベルコンフィグレーションファイルに CALLTESTCONFIG オプションを設定しなければなりません。このオプションは呼出し規約テスター用のコンフィグレーションをもつファイルのパスを指定するもので、デフォルトでは TARGET ディレクトリにある calltest.cfg を使用するようになっています。

```
CALLTESTCONFIG=calltest.cfg
```

テスト群は LOG/ディレクトリのサブディレクトリ gendir/に生成されます。パスしたテストはテストドライバが削除し、フェイルしたテストだけが残ります。

10.2.1 2 つのコンパイラによる呼出し規約テスターの使用

呼出し規約テスターの実行では、一つのコンパイラで一方のファイルをコンパイルし、別のコンパイラで他方のファイルをコンパイルするように設定することができます。この設定により、リンクして実行するときに双方のコンパイラが同じ呼出し規約に従っていることを検証します。2 つのコンパイラで呼出し規約テスターを使用するコンフィグレーションの例が、SuperTest/SuperGuard のインストールディレクトリのサブディレクトリ configs/calltest/ にあります[2]。

このコンフィグレーションファイル例では、コンパイラを設定する通常の CC と CFLAGS オプションに加えて、2 つ目のコンパイラを設定する CC2 と CFLAGS2 オプションを使用します。別に、LINKER オプションで、生成されるオブジェクトファイルをリンクするためのリンカコマンドを設定します。これは、CC や CC2 オプションと同様か、よりリンカコマンドに固有なものになります。

コンフィグレーション例の stplugins.py スクリプトでは、compileAndLink モジュールを改造して、呼出し元を含むファイルを CC オプションで設定されたコンパイラを使用してコンパイルし、CC2 オプションによるコンパイラ設定を使用して呼出し先を含むファイルをコンパイルします。診断ライブラリのソースのような他のファイルはすべて、CC オプションで設定されたコンパイラでコンパイルされます。

10.3 ランダムジェネレータのシードの設定

呼出し規約テスターは疑似ランダムジェネレータを使用して、型を選択し、その型に代入する乱数値を生成します。疑似ランダムジェネレータが前のものと同じシードで開始され、その他の点でコンフィグレーションが同じであれば、同じ関数プロトタイプと値が生成されます。

ランダムジェネレータのシードは strun テストドライバの LOG ディレクトリの calltest.seed ファイルで指定されます。ジェネレータ開始時にこのファイルが存在せず、コンフィグレーションファイルで random_seed オプションが設定されている場合は、それで設定されたシードを使用します。そうでない場合は、タイムスタンプを使ってシードを指定します。シードの書式は以下の例のように、空白で区切られた 4 つの整数です。

```
1641374533 46896415 328274906 1641359533
```

呼出し規約テスターが正常に終了すると、次回の実行で異なるファイルが生成されるように calltest.seed が更新されます。

生成されたテストファイルそれぞれで、乱数シードはファイルの先頭のコメントに記載され、テストファイルを再生成できます。それは、コンフィグレーションファイルの random_seed オプションを使用するか、そのシードを strun テストドライバの LOG ディレクトリの

[2] SuperTest デモ版には呼出し規約テスターの設定例や stplugins.py の変更例は含まれていません。

calltest.seed ファイルに置くことかいずれかで行えます。スタートアップ時に strun はデフォルトで LOG ディレクトリを空にすることに注意してください。これを防ぐには、strun に --keep オプションを渡します。

10.4 完全なテスト集合の生成

デフォルトでは、呼出し規約テスターはコンフィグレーションオプション max_test で指定される数を限度としてテストをランダムに生成しますが、可能なテストファイルの集合すべてを生成することもできます。変数や引数に代入される数値はランダムですが、プログラムの型と形状は完全に数え上げられます。そうするためにはコンフィグレーションファイルで次のオプションを指定します：

```
generate_random = 0
```

これで、テストファイルはランダムではなく順番に生成されます。生成されるテストの数は、設定オプション max_test で指定される数を限度としますが、max_test を -1 と設定することでジェネレータはテストケースを数え上げて行き、すべての可能なテストファイルが生成されると停止します。これを行う場合、非常に制限されたコンフィグレーション（たとえば、new_mask オプションで型を 1 つか 2 つ使用するだけなど）で開始して、生成されるファイルの数を妥当な限度に抑えることを推奨します。

呼出し規約テスターの実行例

以下、Windows 10 で gcc を使用した場合の呼出し規約テスターの実行例を示します。

SuperTest デモ版の実行例では、あらかじめ生成された呼出し規約用のテストプログラム（テストスイートフォルダ内の `calltest` フォルダに置かれています）を実行するだけであり、以下のようにテストリスト内で「`calltest`」として指定します。

```
# TEST          PROPERTIES

2/2/1/1/t100.c  [suffix(2)]
    ⟨

C99/6/8/6/4/t2.c        [suffix(C99)]

# Add these if the compiler can switch to C++ mode based on the suffix,
# and link to a library that is compiled in C mode.
#
#iso14882/18/2/1/t01.C          [suffix(Cpp)]
#iso14882/23/3/2/1/t01.C        [suffix(Cpp)]
calltest
```

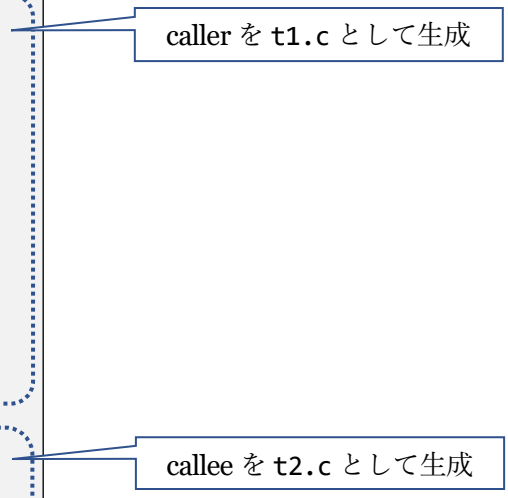
先（p.1）に『呼出し規約テスターは、C 言語のソースファイルのペアを生成します。このペアの一方（呼び元）にはランダムに生成された値を実パラメータに設定した関数の呼出しがあり、もう一方（呼び先）には呼び出される関数の実装があって、入力となる仮パラメータが予期される値であるかどうかをチェックします。さらに、その関数はランダムに生成した値を返し、その戻り値が呼び元のファイルにある呼出し元でチェックされます』とありますが、以下のような処理が行われます。

呼出し規約のテストプログラムは次のように `test.gen` のような名称になっています。`.gen` はジェネレータで生成されるテストプログラムを表す拡張子です。この `.gen` ファイルはスクリプトとして解釈され、先頭行の `#!cvalsplit` によって、（caller と callee を分割する）cvalsplit プログラムでこのファイルを処理し、テストファイルの対をワークディレクトリに作成します。

`@CVALCOMPFILE` でコンパイラに渡すファイルが指定されます。この例では、`t1.c` と `t2.c` になり、`t1.c` が caller で `t2.c` が callee になります。

test.gen ファイル

```
#!cvalsplit

@CVALCOMPFILE t1.c
#include "def.h"

typedef union {
  signed char v0[2];
} t0;

MAIN
{
  t0 v0;
  v0.v0[0] = -47;
  func(v0);
}

@CVALCOMPFILE t2.c
#include "def.h"

typedef union {
  signed char v0[2];
} t0;

extern void func(t0 v0);

void func(t0 v0)
{
  CVAL_VERIFY(v0.v0[0] == -47);
  CVAL_VERIFY(v0.v0[1] == -102);
  v0.v0[1] = -102;
  func(v0);
}
```



GUI での実行は、Execute All Test をクリックすることで同じ結果が得られます。

テストで calltest が実行されると、SuperTest 実行の環境でこの 2 つのソース t1.c と t2.c がそれぞれコンパイル・リンクされて実行されます。

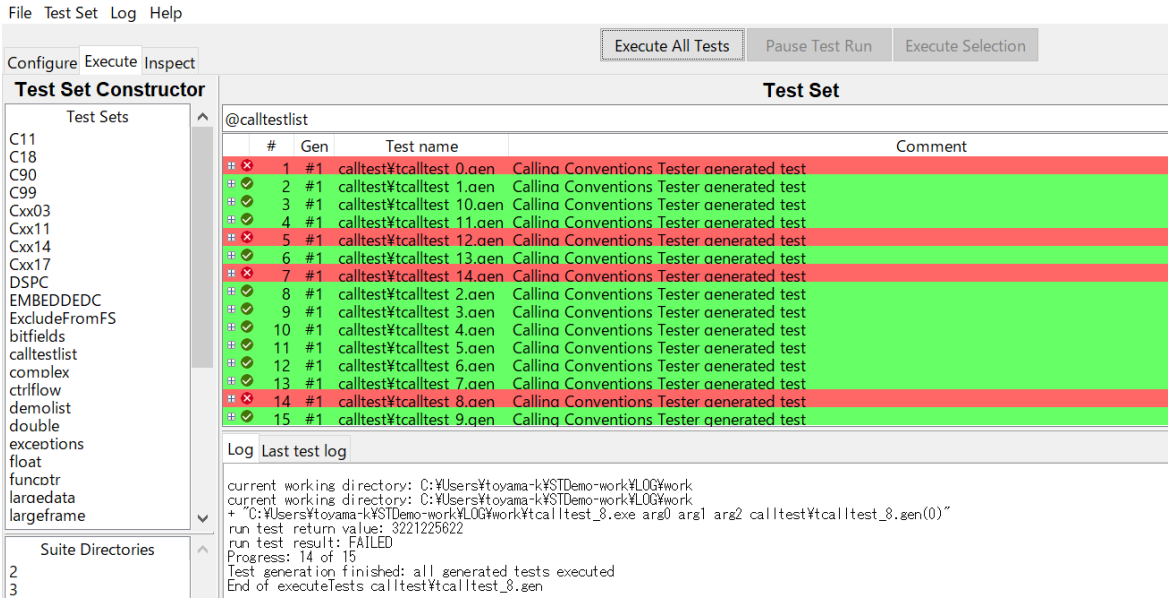
コンパイラとオプションの指定は、.cfg での CC と CFLAGS で行いますので、caller、callee とも同じコンパイラの同じコンパイルオプションが適用されることになります。

別オプションや別コンパイラでのテストを実行するため、.cfg で CC2 および CFLAGS2 を指定することによって、callee を CC2 コンパイラの CFLAGS2 オプションによってコンパイルし、さらに LINKER で指定されたリンカでリンクしてテスト実行できます。(デモ版の SuperTest には CC2 や CFLAGS2、LINKER を反映するスクリプトが提供されていません)

コンパイラのバージョン違いで呼出し規約のエラーが検出される実行例を示します。この例は gcc のバージョン違いのために発生するものです。

呼出し元（caller）：MinGW（gcc 10.1）

呼出し先（callee）：MinGW（gcc 3.2）



なお、呼出し関係を逆（caller が gcc 3.2、callee が gcc 10.1）にした場合、このエラーは出ません。

FUJI SETSUBI

富士設備工業株式会社 電子機器事業部　www.fuji-setsu.co.jp