Features in Use - *From Learning to Test:Accelerating C Compiler Testing*

http://l2t.github.io

There are four groups of features that used in accelerating C compiler testing.

Language Features are concerned with the basic elements of a C program from the perspective of programming language. Intuitively, some bugs only occur on some specific programming elements, and thus the existence of these programming elements can serve as features to characterize test programs triggering bugs. For example, bugs in loop optimization occur when test programs have loop statements. Based on this intuition, our approach (LET) considers the following language features:

- a set of variable features, each of which refers to whether a program contains some type of variables.

  > e.g., boolean, char, float, integer, array, complex, enumeral.

- a set of operator features, each of which refers to whether a program contains some operators.

  > e.g., arithmetic operators, relation operators, assignment operation, logical operators.

- a set of statement features, each of which refers to whether a program contains some type of statements.

  > e.g., break, continue, case, do, for, goto, if, label, switch, return, null, while.

- a set of struct related features, each of which refers to the number of structs with non-zero, zero, const, volatile, or full bitfields of a program.

> - *bitfield:*
>   They can only be declared inside a structure or a union, and allow you to specify some very small objects of a given number of bits in length. For example,
>
>   ```
>   struct {
>   /* field 4 bits wide */
>   unsigned field1 :4;
>   ...
>   };
>   ```
>
> - *volatile:*
>   In computer programming, particularly in the C, C++, C#, and Java programming languages, the volatile keyword indicates that a value may change between different accesses, even if it does not appear to be modified. This keyword prevents an optimizing compiler from optimizing away subsequent reads or writes and thus incorrectly reusing a stale value or omitting writes.
>
> -- from WIKIPEDIA

- a set of pointer related features, including the number of pointers pointing to pointers, the number of pointers pointing to scalars, or the number of pointers pointing to structs, and the percentage of pointers have NULL in alias set.

> - *Alias:*
>   In computing, aliasing describes a situation in which a data location in memory can be accessed through different symbolic names in the program.

Size features are concerned with the program size. Intuitively, the larger the size of a program is, the more likely this program triggers bugs.

- scale, which refers to the number of statements of a program.
- alias size, which refers to the average size of all alias sets of a program.

- a set of structural depth features, each of which refers to the max depth of structs, expression, block, or dereference of a program.

    > - *Dereference*:
    > The dereference operator or indirection operator, denoted by "*", is a unary operator found in C-like languages that include pointer variables. It operates on a pointer variable, and returns an l-value equivalent to the value at the pointer address. This is called "dereferencing" the pointer. For example: > int x;
    > int *p; // * is used in the declaration:
    > // p is a pointer to an integer, since (after dereferencing),
    > // *p is an integer
    > x = 0;
    > // now x == 0
    > p = &x; // & takes the address of x
    > // now p == &x, so *p == x
    > *p = 1; // equivalent to x = 1, since *p == x
    > // now *p == 1 and *p == x, so x == 1
    >
    > -- from WIKIPEDIA

Complexity features are concerned with program complexity. Intuitively, the more complex a program is, the more likely this program triggers bugs. Moreover, the various combinations of complexity features of a test program may have influence on triggering bugs.

- a set of address features, each of which refers to the number of times the address of a struct or a variable is taken respectively.

- a set of struct bitfield features, which refers to the times of a struct with bitfields on LHS and the times of a struct with bitfields on RHS.

    > - *LHS & RHS*:
    > In mathematics, LHS is informal shorthand for the left-hand side of an equation. Similarly, RHS is the right-hand side. The two sides have

> the same value, expressed differently, since equality is symmetric.
>
> -- from WIKIPEDIA

- a set of single bitfield features, which refers to the times of a single bitfield on LHS and the times of a single bitfield on RHS.
- a set of pointer dereference features, which refers to the times of a pointer is dereferenced on LHS and the times of a pointer is dereferenced on RHS.
- a set of pointer comparison features, each of which refers to the number of times a pointer is compared with NULL, with the address of another variable, or with another pointer, respectively.
- times of pointers, which refers to how many times of a pointer is qualified to be dereferenced.
- a set of jump features, which refer to the times of forward jumps and the times of backward jumps.
- a set of used variable features, which refer to the percentage of a fresh-made variable (i.e. the variable defined and used in the same statement) is used and the percentage of an existing variable is used.

Rarity features mean that developers and testers hardly write programs containing these features. Therefore, the parts of compilers related to these features may be tested insufficiently. Intuitively, if a test program contains these features, the test program is more likely to detect bugs related to those parts of the compiler under test.

- a set of volatile access features, which refer to the number of times a non-volatile variable is read, the number of times a non-volatile variable is written, the number of times a volatile variable is read, and the number of times a volatile variable is written.

> Volatile keyword is quite rare to be used in c/c++. But in java or c#, it is a

> property of a variable and indicates that the object to which the variable is bound may mutate, and is specifically intended for threading.
>
> -- from WIKIPEDIA

- a set of specific volatile access features, which refers to the number of times a non-volatile variable reads through a pointer, the number of times a non-volatile variable writes through a pointer, the number of times a volatile variable reads through a pointer, and the number of times a volatile variable writes through a pointer.
- a set of access availability features, which refers to the times of a volatile variable is available for access and the percentage of access of non-volatile variables.