Functionnality of Forester

Cédric Vernou
Programmeur émérite

22 juillet 2013

Table des matières

1 Introduction

In my school project, I have worked on Forester. Currently, Forester don't have documentation and I have exchanged a lot of email with its authors. This document is a synthesis of this explanation and my experimentations.

Forester is an experimental tool for checking manipulation of dynamic data structures using *Forest Automata*. This tool can be loaded into *GCC* as a *plug-in*. This way you can easily analyse C code sources, using the existing build system, without manually preprocessing them first. However, the analysis itself is not yet mature enough to be able to verify ordinary programs.

2 Use Forester

2.1 To use

Forester is plugin GCC, you can use this command to load plugin in GCC and check code source :
gcc -fplugin=libfa.so

But Forester is provided with some scripts for ease of use. In the first place, you need to configurate some variables in console with this script "./fa_build/register-paths.sh". You can use this command :
. ./fa_build/register-paths.sh

This script add paths in your terminal toward other scripts to execute easily Forester. Now, you can use Forester scripts to execute GCC and to load Forester. The main script is :
fagcc main.c

2.2 Some remarks

This scripts are really easy to use, but they have a flaw. You can only check one file at a time.
In connection with this flaw the only file in parameter shall contain the main method. Else Forester returns you this error :

```
error: main() not found at global scope
```

A method to check your files : it is using an other forester.c file and include all files. In forester.c, write main method where you call all methods you want test.

```c
        #include "list.c"

        int main()
        {
                Node * list = initializeList(1);
                finalizeList(list);
                return 0;
        }
```

3 Forester fonctionnalities

3.1 Type primitif

The manage of primitif type in Forester is not implemented yet. If you use pointer on primitif type, Forester returns this error :

```c
        int * integer = malloc(sizeof(int));
```

Forester return this resultat :

```
error: BoxMan::getTypeInfo(): type for int not found!
```

The reason is primitif types have different representation in GCC and so Forester cans not recognize it. This fonctionnality is comming soon.

But presentely, you can get around this problem with this solution :

```c
        struct T { int n; }* p_int;
        p_int = malloc(sizeof(*p_int));
        p_int->n = 10;
        free(p_int);
```

3.2 To use no allocated structure

Before using dynamic structure, it needs to be allocated in memory. Forester checks if all dynamic stuctures are correctely allocated before they can be used. This example shows dynamic stucture which are used without allocation :

```
        struct T { int n; }* pmi;
        pmi->n = 10;
```

In this case, Forester return this error :

```
unallocated\_structure.c:7:9: note: #2317:pmi->n = (int)10
unallocated\_structure.c:7:9: error: dereferenced value is not a valid
        reference [(undef)]
```

3.3 Dynamic structure not desallocated

Forester cans check when a dynamic structure, who is correctely allocated, is not desallocated. In this example, pmi is allocated but this structure is not desallocated and the reference is lost after function's end :

```
        struct T { int n; }* pmi;
        pmi = malloc(sizeof(*pmi));
        pmi->n = 10;
        return;
```

In this case, Forester returns this error :

```
error: garbage detected
```

3.4 Reference to dynamic structure no catch in return of function

Sometime, a function allocates a dynamic structure and returns the reference. In this case, it is important to catch this reference in the aim of desallocating this dynamic structure. Forester checks if all references returned by function are correctely caught.

```c
struct T { int n; };
typedef struct T T;

T * foe()
{
        return malloc(sizeof(T));
}

int main()
{
        foe();
        return 0;
}
```

In this case, Forester returns this error :

```
error: assertion failed
```

This fonctionnality is user-friendly to avoid memory leaks. But it is possible function returns one reference to structure which are not allocated in this function. Forester returns the same error when this return is not caught.

```c
struct T { int n; };
typedef struct T T;

T * foe(T * t)
{
        return t;
}

int main()
{
        T t;
        foe(&t);
        return 0;
}
```

Glossary

Forest Automata Tuples of tree automata which accept trees whose leaves can refer to the roots of all trees accepted by these automata. 2