CPF 软件用户使用手册

二零一四年十一月二十日

目录

1 引言

CPF [UNITS] 工具提供了针对 C 语言的单位信息错误检测，用户通过在源代码中，对变量或函数添加特定注释的方式，完成对单位信息的检测工作。

本工具依赖于 CPF（C Policy Framework）框架，后者重写了 C 语言在“重写逻辑”上的解释，并使用 Madue 引擎对源代码进行解释。所以 CPF [UNITS] 工具可分为：

1）前端工作，主要由 cil 配合完成，抽取出待检测任务

2）后端，对任务文件使用重写逻辑引擎分析，给出验证结果。

本使用手册，向用户介绍了 CPF [UNITS] 的运行环境、软件功能、软件安装以及使用方法等。

参考资料

CPF 项目主页：

http://fsl.cs.illinois.edu/index.php/C_Policy_Framework

CPF[UNITS] GitHub 项目地址：

https://github.com/ice-tea/CPF-UNITS

2 简介

CPF[UNITS]依托于 UIUC 的 C Policy Framework 项目， 提供了针对 C 语言特性的验证平台，主要包含指针别名验证，和单位信息验证两个重要实例。

对单位信息的验证过程，需要先对待测代码的量纲信息写入到源码中，以特殊的注释方式，我们将在后面的实例中做详细介绍。

3 运行环境

3.1 硬件要求

CPF 对内存和 CPU 的要求不高，绝大多数计算机都可以运行。

3.2 支持的操作系统

CPF[UNITS]支持 Linux 平台。需要下载源码编译。

4 编译安装（Linux 用户）

4.1 下载源码

CPF 在 GitHub 上提供源代码下载。可以使用 git 工具：

```
$ git clone https://github.com/ice-tea/CPF-UNITS.git
```

4.2 准备工作

下载好源代码后，可以使用各类最新的 C++编译器（支持或部分支持 C++11）进行编译。除此之外，本工具前端使用 CIL 工具包，其需要安装 OCaml，这里可以下载 3.8 版本的 OCaml。为了用户方便，我们已在工具包的 build / ocaml-3.08.2 目录下，包含了 OCaml 的安装包，用户只需要如下命令即可完成 OCaml 安装：

```
$ cd ./bulid/ocaml-3.08.2
$ ./configure
$ make world
$ make opt
$ make install
```

Ocmal 的安装完毕便可以继续下一步工具配置。

4.3 编译代码

前端 CIL 需要编译，过程如下

```
$ cd ./bulid/cil
$ ./configure
$ make
```

完成如上命令，整个工具的配置就完成；接下来会具体介绍工具的使用方法。

5 测试实例

以下我们介绍 CPF[UNITS]的测试方法。

5.1 第一个例子

假设我们有如下一段简单的代码：

```
$ cat test.C
struct A
{
   int a;
};
int num1 = 8; //@ assume(UNITS): @unit(num1) = $m
int main()
{
   int num2 =10; //@ assume(UNITS): @unit(num2) = $km
   struct A aA;  //@ assume(UNITS): @unit(aA.a) = $km

   if(num1 < num2){ //error1 here
      num1 += aA.a;  //error2 here
   }
   else{
      num2 += aA.a;
   }
   return 0;
}
```

第一步将变量的具体单位信息写入注释，规范如上所示。全局变量 num1 应该是米单位，局部变量 num2 是千米单位，而结构体 aA.a 是千米单位。

添加单位信息，便可以通过本工具对代码的单位信息运算合法性做检测，命令如下

```
$ ./cpf-test test.C
```

工具运行结果如下：

```
result StringList: "Function main: ","ERROR on line 14(1): Unit violation
detected in less than operation, incompatible units.","ERROR on line
15(1): Unit violation detected in addition operation, incompatible
units.",
```

5.2 函数注释

CPF[UNITS]同样提供了对函数的注释方法，可以通过前置、后置条件加以说明；如下被测函数，接收两个千米单位，返回其和，并转换为米单位：

```
int km2m(int input2, int input2){
        int output = input1 + input2;
        return output*1000;
}
```

对其注释可以如下：

```
//@ precondition(UNITS): @unit(input1) = $km
//@ precondition(UNITS): @unit(input2) = @unit(input1)
//@ postcondition(UNITS): @unit(@result) = $m
```

不仅支持对特定单位的注释，还可以表达参数间的单位信息关系。可见工具提供了丰富的表示方式，可以满足函数测试需求。

6 测试结果

6.1 函数测试结果

这里将上一节函数测试实例完整化：

被测代码：

```
//@ precondition(UNITS): @unit(input1) = $km
//@ precondition(UNITS): @unit(input2) = @unit(input1)
//@ postcondition(UNITS): @unit(@result) = $m
int km2m(int input1, int input2){
   return (input1+input2)*1000;
}

int main()
{
   int num1 =1; //@ assume(UNITS): @unit(num1) = $m
   int num2 =2; //@ assume(UNITS): @unit(num2) = $km

   km2m(num2,num2); //right
   km2m(num1,num2); //wrong: input should be km

   num1 += km2m(num2,num2); //right
   num2 = num2 + km2m(num2,num2);  //wrong: result is m, can't add km
   return 0;
}
```

工具测试结果如下所示：

```
result StringList: "Function main: ","ERROR on line 17(1): Assert
failed!","ERROR on line 20(1): Unit violation detected in addition
operation, incompatible units.","Environments created = 1"
```