FITR304: Project

Author : Christophe Garion <garion@isae.fr>
Audience : IN
Date : 01/29/17

Abstract

The aim of this mini-project is to specify, develop and prove a mini-library for string manipulation in C using the WP Frama-C plugin [2].

1 Problem

We want here to develop a mini-library in C for string manipulation. Three functions have to be specified using ACSL, developed in C and proved using the Jessie plugin of Frama-C:

- `int strlen(cons char *str)`: function that returns the length of the `str` string
- `void strsubstring(char *dst, const char *src, int start, int length)`: function that stores into `dst` the substring obtained from `src` by selecting characters from index `start` to `start + length - 1`
- `void strappend(char *dst, const char *src)`: function that appends string `src` to string `dst`

2 Working with pointers and arrays

You will find detailed explanations in [1]. The following constructs are useful[1]:

- `\valid(p)` (where `p` is some term of pointer type) holds if and only if dereferencing `p` is safe. It can also be applied to set of terms of some pointer type.
- `\valid_range(p,a,b)` (where `p` is some term of pointer type and `a` and `b` two integers) holds if and only if dereferencing `p+a`, …, `p+b` is safe. It is in fact equivalent to `\valid(p+(a..b))`.

You can specify that two memory regions do not overlap with the `\separated` predicate. For instance, `\separated(p+(0..3), q+(1..2))` specify that the region starting from `p` and finishing at `p+3` does not overlap with the region starting from `q+1` and finishing at `q+2`. See [1] for more details.

3 Mathematical definitions and files in repository

You will find three files in your personal repository:

- `include/my_string.h`: the specification of the three functions
- `tests/my_string_tests.c`: complete Unity tests for the three functions
- `Makefile`: a classical makefile with rules for proving and testing your implementation

A mathematical definition of the length of a string is given as a function in `my_string.h`:

```
/*@
  @ axiomatic string {
  @  logic integer strlen_{L}(char *src)
  @    reads src[0..];
  @  axiom strlen_inside{L}:
  @    \forall char *src; \forall integer x; 0 <= x < strlen_(src) ==> src[x] != '\0';
  @  axiom strlen_end{L}:
  @    \forall char *src; src[strlen_(src)] == '\0';
  @  axiom strlen_pos{L}:
  @    \forall char *src; strlen_(src) >= 0;
  @ }
  @*/
```

`strlen_` can be used with an optional label (e.g. `strlen_{LoopEntry}(s)`).

[1] Those constructs can be used with labels like the `\at` construct, cf. [1].

4 Some advices

- write simple code: for instance, do not try to use pointer arithmetics to iterate in a loop, rather use a separate integer.
- do not try to verify properties on the length of the strings in postcondition. The given mathematical definition is not sufficient to prove them.
- when specifying separated regions, you can use in a first time regions like `p+(..)` which is clearly an over-approximation.

5 Grading

For each of the functions you have to develop and specify, you will be graded on:

- the functional code you have written
- the completeness of your specifications
- the correctness of your specifications

6 Deliverables

All developed or completed C source files should be annotated and put into the `src` directory of your Subversion project repository that is located at `https://eduforge.isae.fr/repos/FITR304/deductive/common` . You will also provide in the repository a `README.txt` file (`txt` format) explaining the assertions you have written. Use the line numbers in your source file, e.g. "precondition at line 6 means that …".
Final version of your files should be committed before February 26th 2017 at 11:00 PM.

References

[1] Patrick Baudin et al. *ACSL: ANSI/ISO C Specification Language*. Version 1.12. 2016. URL: `http://frama-c.com/download/acsl-implementation-Silicon-20161101.pdf`.

[2] Patrick Baudin et al. *WP Plug-in Manual*. 2016. URL: `http://frama-c.com/download/wp-manual-Silicon-20161101.pdf`.