CO322: DS & A
Note on correctness

Dhammika Elkaduwe

Department of Computer Engineering
Faculty of Engineering

University of Peradeniya

Correctness of an algorithm

Take bubble sort, for example,

- Question: How fast is this algorithm? (≈complexity)
- Question: does this algorithm work? (correctness)
- Are they the same?

Correctness of an algorithm

Take bubble sort, for example,

- Question: How fast is this algorithm? ($\approx$complexity)
- Question: does this algorithm work? (correctness)
- Are they the same?

Correctness: Bit formal definition

The algorithm is correct if for any **valid** input it produces the **expected result**

Two points:

- valid input (it does not have to work for all inputs)
- expected result (might not be clear in all cases)

Sample code (L4Ka::Pistachio V4)

Code fragment from *schedule*. Written in C++

```
/**
 * selects the next runnable thread and activates it.
 * @return true if a runnable thread was found, false
     otherwise
 */
bool scheduler_t::schedule(tcb_t * current)
{
    tcb_t * tcb = find_next_thread (&root_prio_queue);

    ASSERT(tcb);
    ASSERT(current);

    // the newly selected thread gets accounted
    get_prio_queue(tcb)->timeslice_tcb = tcb;
```

Code cannot work with a NULL *tcb* or a *current*

Sample code (L4Ka::Pistachio V4)

Neat trick in coding.

```
#if !defined(CONFIG_KDB_NO_ASSERTS)
# define ASSERT(x) \
do { \
    if (EXPECT_FALSE(! (x))) { \
        printf ("Assertion "#x" failed in file %s, line %d
            (fn=%p)\n",\
                __FILE__, __LINE__,
                    __builtin_return_address(0)); \
        enter_kdebug ("assert");\
    }\
} while(false)
# else /* defined(CONFIG_KDB_NO_ASSERTS) */
#define ASSERT(x)
```

Idea: ASSERT works only when debugging the code. Removed in the production code! (why: takes time)

Homework

Answer the following questions:

- Is it good idea to remove asserts?
- Should assert be replaced with *if*?
- What about assert in Java?
- What is the different between throwing an exception and assert?

Correctness: Bit formal definition

The algorithm is correct if for any **valid** input it produces the **expected result**

Expected result:

- In some cases it might be clear
- At times difficult to specify (clearly say)

Correctness: formal definition

Given the precondition a correct algorithm will guarantee the post-condition.

Oracle documentation on using pre- and post-conditions in Java:
https://docs.oracle.com/cd/E19683-01/806-7930/assert-13/index.html

How about sorting algorithms

Take bubble sort, what is the:

- precondition? Items should be comparable
- postcondition? $\forall i, data[i] \geq data[i-1]$

How about sorting algorithms

Take bubble sort, what is the:

- precondition? Items should be comparable
- postcondition? $\forall i, data[i] \geq data[i-1]$

How about sorting algorithms

Take bubble sort, what is the:

- precondition? Items should be comparable
- postcondition? $\forall i, data[i] \geq data[i-1]$

Verifying the correctness of algorithms

We can use:

- Testing (problem of *test coverage*)
- Using asserting and invariants
- Model checking (model of the program: ex: finite state machines)
- Correct by design
- Formal verification

What should I use? Depends on what you are building (example: calculate your age given BD vs. calculate force to apply on break pads)

Verifying the correctness of algorithms

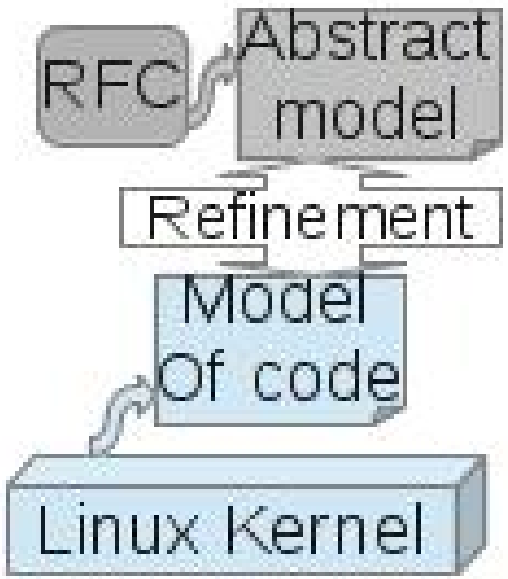
We can use:

- Testing (problem of *test coverage*)
- Using asserting and invariants
- Model checking (model of the program: ex: finite state machines)
- Correct by design
- Formal verification

What should I use? Depends on what you are building (example: calculate your age given BD vs. calculate force to apply on break pads)

At home

Is the Linux TCP stack correct?



- Derive a model from the Linux kernel code
- Derive a model from the RFC
- Prove properties at the abstract model
- Show that two models are same via *refinement*