

Computer Systems and Correctness

Suppose we design a (complex) computer system, which may contain various components, for example, hardware, software etc. We have high requirements to the **correctness** of the system (**safety, reliability, security, consistent state, no deadlocks** etc.)

How can one achieve a **100% safety**?

Computer systems are becoming increasingly unreliable.

Small Example: Software

Consider the following fragment of a C++ program:

```
int* allocateArray(int length)
{
    int* array = malloc(sizeof(int)*length);
    int i;
    for (i = 1; i <= length; i++) {
        array[i] = 0;
    }
    return array;
}
```

Is this program correct?

Small Example: Software

Hardly: it writes into memory that has not been allocated:

```
int* allocateArray(int length)
{
    int* array = malloc(sizeof(int)*length);
    int i;
    for (i = 1; i <= length; i++) {
        array[i] = 0;
    }
    return array;
}
```

Small Example: Software

```
int* allocateArray(int length)
{
    int* array = malloc(sizeof(int)*length);
    int i;
    for (i = 1; i < length; i++) {
        array[i] = 0;
    }
    return array;
}
```

Is the program correct now?

Small Example: Software

No: it may write to the null address:

```
int* allocateArray(int length)
{
    // call to malloc below may return 0
    int* array = malloc(sizeof(int)*length);
    for (int i = 1; i < length; i++) {
        array[i] = 0;
    }
    return array;
}
```

Small Example: Software

Let's change it:

```
int* allocateArray(int length)
{
    int* array = malloc(sizeof(int)*length);
    if (! array)
        return 0;
    for (int i = 1; i < length; i++) {
        array[i] = 0;
    }
    return array;
}
```

Is the program correct now?

Small Example: Software

```
/* Returns a new array of integers of a given
   length initialised by a non-zero value */
int* allocateArray(int length)
{
    int* array = malloc(sizeof(int)*length);
    if (! array)
        return 0;
    int i;
    for (i = 1; i < length; i++) {
        array[i] = 0;
    }
    return array;
}
```

Is it correct now?

The big question

We discussed **correctness** of a program without ever defining what it means.

So what is correctness?

Notes

Note:

- ▶ We could spot the first two errors without knowing anything about the **intended meaning** of the program. But we had to understand the meaning of C programs in general and some specific properties of programming in C.
- ▶ To understand the last “error” we had to know something about the **the intended behaviour of the program**.

Back to the Original Program

```
/* Write a program that crashes */
int* allocateArray(int length)
{
    int* array = malloc(sizeof(int)*length);
    int i;
    for (i = 1; i <= length; i++) {
        array[i] = 0;
    }
    return array;
}
```

Another example: circuit design

We used a circuit C_1 in a processor and would like to replace it by another circuit C_2 . For example, we may believe that the use of C_2 results in a lower energy consumption.

We want **to be sure** that C_2 is correct, that is, it will behave according to some specification.

If we know that C_1 is correct, it is sufficient to **prove** that C_2 is functionally equivalent to C_1 .

Another example: Vending Machine

1. The vending machine contains a drink storage, a coin slot, and a drink dispenser. The drink storage stores drinks of two kinds: beer and coffee. We are only interested in whether a particular kind of drink is currently being stored or not, but not interested in the amount of it.
2. The coin slot can accommodate up to three coins.
3. The drink dispenser can store at most one drink. If it contains a drink, this drink should be removed before the next one can be dispensed.
4. A can of beer costs two coins. A cup of coffee costs one coin.
5. There are two kinds of customers: students and professors. Students drink only beer, professors drink only coffee.
6. From time to time the drink storage can be recharged.

Another example: Vending Machine

Suppose that we would like to **derive** some properties of the described vending machine model, for example that a student would never leave money in the coin slot.

How to establish correctness

- ▶ Build a **formal model** of the system;
- ▶ Find a **language** for expressing intended properties;
- ▶ The language must have a **semantics** that explains which models satisfy properties expressible in the language.
- ▶ Write a **specification**, that is, intended properties of the system in this language.
- ▶ We must be able to **prove** formally that the system model is also a model of the specification.

What is logic?

- ▷ Syntax and semantics;
- ▷ Proof theory and model theory;
- ▷ Reasoning.

Logic in computer science

- ▶ knowledge representation and reasoning;
- ▶ semantic Web;
- ▶ circuit design;
- ▶ constraint satisfaction;
- ▶ planning;
- ▶ software and hardware verification;
- ▶ databases (semantics and query optimisation);
- ▶ theorem proving in mathematics.

This course

- ▶ propositional logic;
- ▶ satisfiability checking in propositional logic;
- ▶ semantic tableaux;
- ▶ binary decision diagrams (BDDs);
- ▶ quantified boolean formulas;
- ▶ propositional logic of finite domains;
- ▶ state-changing systems and transition systems;
- ▶ temporal logic;
- ▶ model checking.