Computing Fundamentals

Salvatore Filippone

salvatore.filippone@uniroma2.it

2013–2014

Octave/Matlab:

```
fprintf('Hello world\n');
```

Octave entities:

1. Numeric constants;
2. Variables;
3. Operators;

A variable is fundamentally a tag associated with a memory area: it contains *data*. A variable *name* is a sequence of letters and numbers starting with a letter.
Assignment operator:

```
one=1;
```

DataType: A set of entities *and* the operations defined on them.

Examples of *primitive* data types:

- Booleans (logical);
- Integers;
- Floating-point;
- Complex.

In Octave/Matlab: distinction among the numerics is blurred.

Syntactic elements: arithmetic operators

- + Addition;
- - Subtraction;
- * Multiplication;
- / Division;
- ^ Exponentiation;

Octave basics

Syntactic elements: relational operators

- < Less than;
- <= Less than or equal to;
- > Greater than;
- >= Greater than or equal to;
- == Equal;
- ~= Not equal; in Octave (but not in Matlab) also !=

Syntactic elements: logical operators

- `~`, `not` NOT;
- `&`, `&&`, `and` AND;
- `|`, `||`, `or` OR;
- `xor` Exclusive OR;
- `false`
- `true`

SCRIPTS: Sequences of instructions stored in a file, so that they can be retrieved and executed

Examples.

Syntactic elements: functions

- `sqrt` Square root;
- `exp` Exponential;
- `log`, `log2`, `log10` Logarithm;
- `sin`, `cos`, `sinh`, `cosh`, `tan`, `asin`, `acos`, `atan` Trigonometric functions
- `airy`, `besselj`, `beta`, `erf`, `gamma`, `legendre` Special functions

Integers are represented with 8, 16, 32 or 64 bits.

Integer operators work “as expected”, except that the operands must be of the same type.

In Octave/Matlab most operations implicitly convert to floating-point.

Since computers represent numbers with *finite* strings of digits (bits), we will never have "real" numbers; what we can aim for is a (finite) subset of the rationals.
Therefore any "real" number x will be represented by an approximation

$$\hat{x} = \mathrm{round}(x) = x(1+\delta).$$

What we can hope is to somehow control the relative error $|\delta| \leq \epsilon$

Floating-point numbers: a subset of the *rational* numbers with the form (s, e, f) where s is the sign, e is the exponent and f is the fraction part, or mantissa; we also have parameters β, t e_{min} and e_{max} such that

$$(s, e, f) = s \times f \times \beta^{e-t} = \pm\beta^{e}\left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots \frac{d_t}{\beta^t}\right)$$

with f represented on t figures.

$$.d_1 d_2 \dots d_t$$

Floating-point numbers: a subset of the *rational* numbers with the form (s, e, f) where s is the sign, e is the exponent and f is the fraction part, or mantissa; we also have parameters β, t e_{min} and e_{max} such that

$$(s, e, f) = s \times f \times \beta^{e-t} = \pm\beta^{e}\left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots \frac{d_t}{\beta^t}\right)$$

with f represented on t figures.

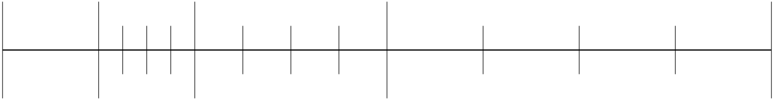
$$.d_1 d_2 \ldots d_t$$

Normalized numbers: $m \geq \beta^{t-1}$, i.e. $d_1 \neq 0$; we also add explicitly the zero (which would be excluded).

Floating-Point Arithmetic

As an example: with $\beta = 2$, $t = 3$, $e_{min} = -1$, $e_{max} = 3$ we can represent the following set of numbers:

$$0, 0.25, 0.3125, 0.3750, 0.4375, 0.5, 0.625, 0.750, 0.875$$
$$1.0, 1.25, 1.50, 1.75, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 6.0, 7.0$$



Note: the relative error decreases within each octave, but the minimum and maximum are the same over all octaves!

If $\beta = 2$ then

$$d_0 \neq 0 \Rightarrow d_0 = 1,$$

therefore the first bit can be assumed ("phantom").

Floating-Point Arithmetic

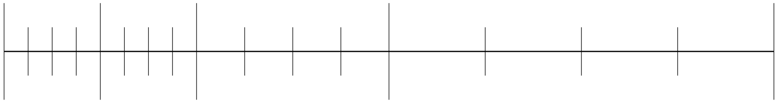
If $\beta = 2$ then

$$d_0 \neq 0 \Rightarrow d_0 = 1,$$

therefore the first bit can be assumed ("phantom").
To get a uniform spacing we introduce the so-called "denormalized" numbers, i.e. numbers with $e = e_{min}, d_0 = 0$; in our example these are

$$0.0625, 0.125, 0.1875,$$

and they fill the "hole" around zero as follows:



Floating-Point Arithmetic IEEE 754

Single precision $\beta = 2$, $t = 23 + 1$ ("phantom" bit is 1 if normalized, 0 otherwise), $e_{min} = -127$, $e_{max} = 127$, $u = 6.0 \times 10^{-8}$, range $10^{\pm 38}$

Double precision $\beta = 2$, $t = 52 + 1$ ("phantom" bit is 1 if normalized, 0 otherwise), $e_{min} = -1023$, $e_{max} = 1023$, $u = 1.1 \times 10^{-16}$, range $10^{\pm 308}$

Extended precision $\beta = 2$, $t = 63 + 1$ ("phantom" bit is 1 if normalized, 0 otherwise), $e_{min} = -16383$, $e_{max} = 16383$, $u = 5.4 \times 10^{-20}$

Quad precision $\beta = 2$, $t = 112 + 1$ ("phantom" bit is 1 if normalized, 0 otherwise), $e_{min} = -16383$, $e_{max} = 16383$, $u = 9.6 \times 10^{-35}$

Octave/Matlab use *double precision* numbers.

When $e = 2047$, the actual exponent is $2047 - 1023 = 1024$; with $f = 0$ this is understood to represent infinity.
When $f \neq 0$, it is a NaN (Not a Number)

- $1/0 = \infty$
- $-1/0 = -\infty$
- $0/0 = \infty - \infty = \infty/\infty = 0 \times \infty = \sqrt{-1} = NaN$

In general a real number a is not exactly representable.
If $\mathrm{fl}(a)$ is the best floating point approximation to a, we define the machine precision through the relation

$$\mathrm{fl}(a) = a(1+\epsilon), \qquad |\epsilon| \leq \epsilon_M \quad \text{for all } a.$$

The quantity ϵ_M is closely related to the number of significant digits, but has nothing to do with the smallest represented number.
In double precision IEEE arithmetic we have $\epsilon \approx 10^{-16}$ whereas the smallest normalized numer is approx. 10^{-308}, and the smallest unnormalized number is approx. 10^{-324}.

Are there any numbers exactly represented in our system?

Are there any numbers exactly represented in our system?
They must have a *finite* expansion:

$$x = \frac{p}{q} = \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots \frac{d_t}{\beta^t} \right)$$

Floating-Point Arithmetic IEEE 754

Are there any numbers exactly represented in our system?
They must have a *finite* expansion:

$$x = \frac{p}{q} = \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots \frac{d_t}{\beta^t} \right)$$

Thus, there exists k such that

$$x \times \beta^k \qquad \text{integer}$$

Are there any numbers exactly represented in our system?
They must have a *finite* expansion:

$$x = \frac{p}{q} = \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots \frac{d_t}{\beta^t}\right)$$

Thus, there exists k such that

$$x \times \beta^k \qquad \text{integer}$$

Therefore we must have

$$\frac{\beta^k}{q} \qquad \text{integer}$$

Floating-Point Arithmetic IEEE 754

Are there any numbers exactly represented in our system?
They must have a *finite* expansion:

$$x = \frac{p}{q} = \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots \frac{d_t}{\beta^t}\right)$$

Thus, there exists k such that

$$x \times \beta^k \qquad \text{integer}$$

Therefore we must have

$$\frac{\beta^k}{q} \qquad \text{integer}$$

but this means that the denominator q only contains prime factors which are also prime factors of β.

Floating-Point Arithmetic IEEE 754

Are there any numbers exactly represented in our system?
They must have a *finite* expansion:

$$x = \frac{p}{q} = \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots \frac{d_t}{\beta^t}\right)$$

Thus, there exists k such that

$$x \times \beta^k \qquad \text{integer}$$

Therefore we must have

$$\frac{\beta^k}{q} \qquad \text{integer}$$

but this means that the denominator q only contains prime factors which are also prime factors of β.
Conclusion: 0.1_{10} is NOT exactly representable in binary arithmetic!!!!

In the analysis of algorithms we will assume that the result of any individual arithmetic operation is the rounding of the exact result:

$$u \oplus v = \mathrm{round}(u + v);$$

This is a requirement of IEEE 754.
Floating-point operations are:

Commutative (where it makes sense)

$$u \oplus v = v \oplus u$$

Non associative

$$(x \oplus y) \oplus z \neq x \oplus (y \oplus z)$$

Non distributive

$$x \otimes (y \oplus z) \neq (x \otimes y) \oplus (x \otimes z)$$