# Computational Paradigms

Work Assignment

# Functional programming

- A polynomial of degree $n$ in the variable $x$ is a function of the form:

$$P(x) = \sum_{i=0}^{n} a_i \cdot x^i$$

- In Scheme, you can represent a polynomial using the list of its coefficients ($a_0$, $a_1$, …, $a_n$)
  - Example: you can represent the polynomial:

    $$6 - 2x - x^3$$

    using the list: `'(6 -2 0 -1)`

# Functional programming

- Define a function `poly-add` that, given the coefficients of two polynomials $P_1(x)$ and $P_2(x)$, returns the coefficients of the polynomial $P_1(x) + P_2(x)$
  - Example:
    `(poly-add '(1 -2 1) '(0 1 0 0 -2))`

should return the coefficients of

$$[1 - 2x + x^2] + [x - 2x^4] = 1 - x + x^2 - 2x^4$$

that are the list `'(1 -1 1 0 -2)`

# Functional programming

- Define a function `poly-mult` that, given the coefficients of two polynomials $P_1(x)$ and $P_2(x)$, returns the coefficients of the polynomial $P_1(x) \cdot P_2(x)$

  - Example:
    `(poly-mult '(1 1) '(1 -1))`

  should return the coefficients of

$$[1 + x] \cdot [1 - x] = 1 - x^2$$

  that are the list `'(1 0 -1)`

# Functional programming

- Define a function `poly-derive` that, given the coefficients of a polynomials $P(x)$ returns the coefficients of the polynomial $dP(x)/dx$

  - Example:
    `(poly-derive '(3 -2 -1 1))`

should return the coefficients of

$$\frac{d(3 - 2x - x^2 + x^3)}{dx} = -2 - 2x + 3x^2$$

that are the list `'(-2 -2 3)`

# Functional programming

- Define a function `poly-solve` that, given the coefficients of a polynomials $P(x)$ returns a solution of the equation $P(x) = 0$.
  - You can use the Netwon method for finding the solution; however, you **must** use the fact that, for a polynomial, you can compute exactly the derivative (see the function `poly-derive` in a previous slide).
    - So you have to modify the `newton` function seen during the lectures

# Functional programming

- Example:
  - Find a solution to the equation:
    $1 - 8x^2 + x^3 = 0$

```
(poly-solve '(1 0 -8 1))
0.3618306551887854
```

# Logic programming

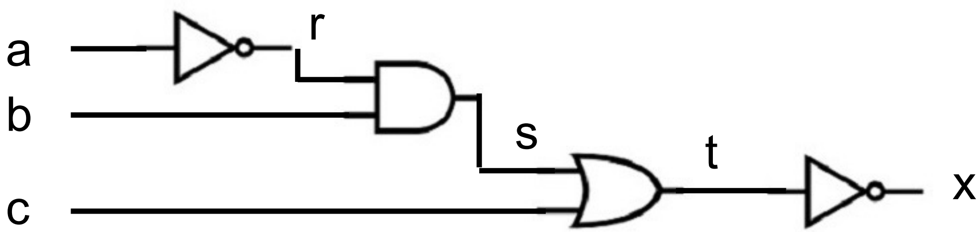
- A logic circuit can be represented using the following facts:

```
% In the following, X, Y and Z are names of signals

not_gate(X,Y)   % there is a NOT gate connecting X to Y
                % (Y is the output)
and_gate(X,Y,Z) % there is an AND gate connecting X
                % and Y to Z (Z is the output)
or_gate(X,Y,Z)  % there is an OR gate connecting X
                % and Y to Z (Z is the output)
```

# Logic programming

- Example: the following circuit:




is represented as:

```
not_gate(a, r).
and_gate(r, b, s).
or_gate(s, c, t).
not_gate(t, x).
```

# Logic programming

- The value of input signals of a circuit can be represented using the fact:

```
input_signal(Signal, Value)
```

- Example: with reference to the previous circuit:

```
input_signal(a, 0).
input_signal(b, 1).
input_signal(c, 0).
```

# Logic programming

- Define a predicate

```
signal_value(Signal, Value)
```

that is true if the signal *Signal* has the value *Value*

- Example: with reference to the previous circuit, the query `signal_value(x, V)` must return:

```
V=0
```

# Logic programming

- The `signal_value` predicate must work correctly also in the case in which some of the input signals have an unspecified value, represented using:

```
input_value(Signal, _)
```

# Logic programming

- Example: given the network:

```
or_gate(a,b,x).
and_gate(c,d,y).
or_gate(x,y,z).
input_signal(a, 1).
input_signal(b, _). % b is not specified
input_signal(c, 1).
input_signal(d, _). % d is not specified
```

the query: `signal_value(z, V)`
must return correctly:

V=1

# Logic programming

- Example: given the network:

```
or_gate(a,b,x).
and_gate(c,d,y).
or_gate(x,y,z).
input_signal(a, _). % a is not specified
input_signal(b, 0).
input_signal(c, 1).
input_signal(d, _). % d is not specified
```

the query: `signal_value(z, V)`

must return correctly both `V=0` and `V=1`