FUNCTIONAL PROGRAMMING

Maths & Natural Sciences (MN) programme at Uppsala University, Sweden

Course Notes by
Pierre Flener, PhD, docent,
IT Dept, Uppsala University, Sweden

Based on the notes of
Prof. Yves Deville,
Université catholique de Louvain, Belgium

Chapter 1: Introduction

(Version of 24 September 2004)

1.1. Objectives

Introduction to the *fundamental* principles and methodologies of functional programming, using the programming language Standard ML (SML, or simply ML) as the teaching medium.

Theoretical focus, with many examples, on:

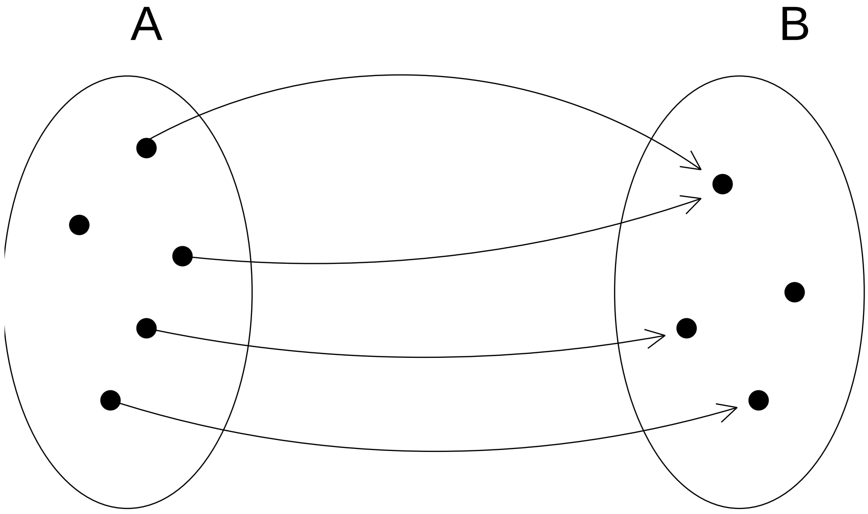
- Algorithms and data structures (**how?**)
- Programming methodology:
  - Importance of *specifications* (**what?**)
  - Importance of *justifications* (**why?**)
  - Importance of other documentation
  - Importance of rigour, explicitness, and elegance
- Complexity of algorithms

Some further *practice* of programming (in ML) is acquired through assignments, which are to be:

1. Prepared at home
2. Tried on the computer in labs under assistant supervision
3. Graded by an assistant

1.2. Functions

A function f is a correspondence between two sets of values:



f : A --> B

To each element a of the set A,
the function f associates at most one value of the set B

Notations

f(a) = b : f associates the value b of B to the element a of A

f(a) = $\perp$ (or f(a) is undefined): f associates no value to a

Total functions and partial functions

Let $\mathsf{f} : \mathsf{A} \rightarrow \mathsf{B}$ be a function:

- f is a *total* function if f is defined for every element of A
- f is a *partial* function if f is not total

Definition of functions

Definition by extension

Give the *graph* of the function: $(a_1, b_1) \quad (a_2, b_2) \quad \ldots$

Example: function double:

(1,2) (2,4) (3,6) (4,8) . . .

Definition by intension (note the '**s**'!)

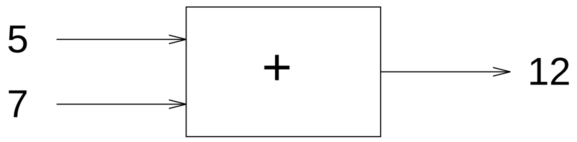
Define the function by a *rule* describing its graph

Example: function double:

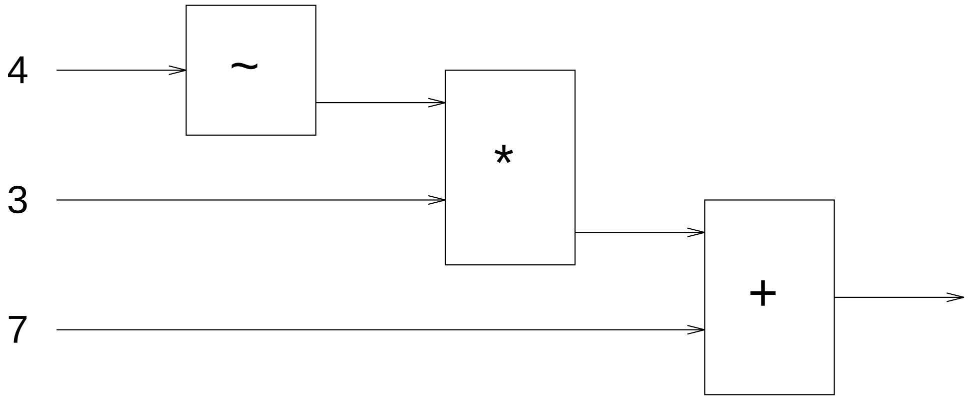
double(n) = 2 ∗ n

Expressions

5 + 7

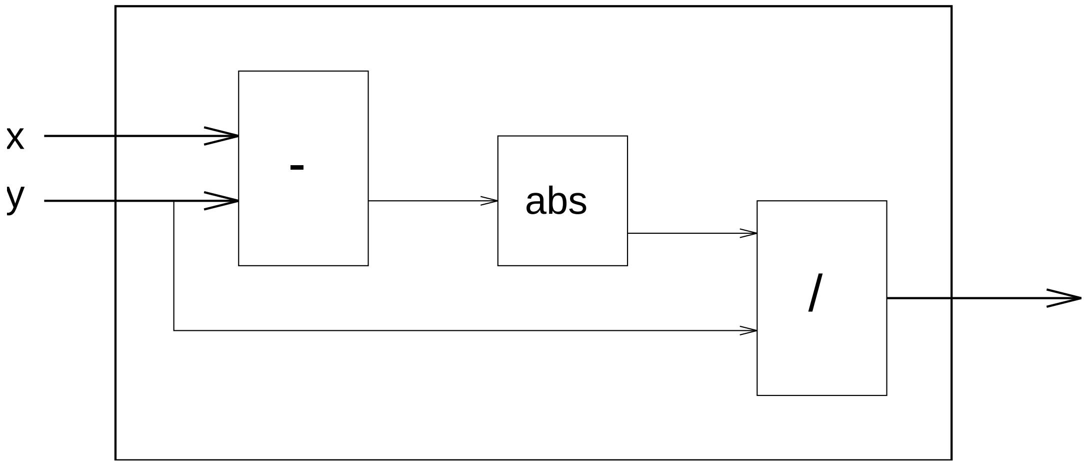


3 * ~ 4 + 7



Definition of new functions

relative_error (x,y) = abs(x - y) / y



1.3. Functional programming languages

Fundamental principles

- Execution by evaluation of expressions
- Declaration of functions
- Application of functions
- Recursion

Existing functional programming languages

- Lisp (Mc Carthy, 1962), Scheme
- FP (J. Backus, 1978)
- Miranda (D. Turner, 1986)
- Haskell (P. Hudack, 1990)
- LCF, ML (Meta Language) (Edinburgh, 1977)
- CAML (France, 1990)
- SML (Standard ML) (1990)