What is Functional Programming?

① Based on first-class, mathematical functions. (inputs → outputs).

- create anonymous functions
- higher-order functions: functions that take other functions as input / return functions as output.

② Declarative: define result you want, not steps to compute it.

→ rewriting / evaluation as model rather than executing steps.

Things that often go with it:

- strong, static type system
- algebraic data types + pattern matching.
- purity, laziness } Haskell-specific.

Rewriting systems

e.g. $2 + ((4 + 9) \times 3)$

→ $2 + (13 \times 3)$

→ $2 + 39$

→ 41

Carry out a sequence of rewriting steps until done.

More generally, a rewriting system consists of a grammar defining the syntax of valid expressions, and rewrite rules that tell us how we are allowed to rewrite expressions.

e.g. $e ::= n$ ← any integer

| $e + e$ ← "or"

| $e - e$

| $e \times e$

We can use parentheses to disambiguate.

e.g. 2

$3 + 17$

$(1 + 2) \times (3 - (4 + 5))$

Rewrite rules:

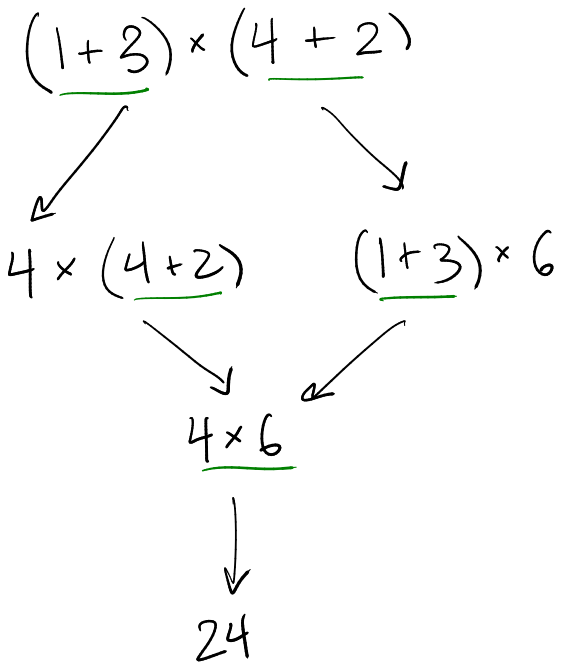
$m + n \longrightarrow$ the integer you get by adding m and n.

e.g. $2 + 3 \longrightarrow 5$.

Similar for $-$, $\times$.

Typically we are allowed to rewrite anywhere inside an expression.

e.g.



Confluence — result of rewriting system does not depend on order of rewrites.

Termination — rewriting process always stops.

Example. (any variable)

$$e ::= x \mid e \wedge e \mid e \vee e \mid \neg e$$

e.g. $(x \wedge y) \vee \neg(y \vee \neg z)$

Rewrite rules: Goal: rewrite to conjunctive normal form.

$\neg(\neg e) \longrightarrow e$

$\neg(e_1 \wedge e_2) \longrightarrow (\neg e_1) \vee (\neg e_2)$

$\neg(e_1 \vee e_2) \longrightarrow (\neg e_1) \wedge (\neg e_2)$

$e_1 \vee (e_2 \wedge e_3) \longrightarrow (e_1 \vee e_2) \wedge (e_1 \vee e_3)$

$(e_2 \wedge e_3) \vee e_1 \longrightarrow (e_2 \vee e_1) \wedge (e_3 \vee e_1)$

$(x \wedge y) \vee \neg(y \vee \neg z)$

$\longrightarrow (x \wedge y) \vee ((\neg y) \wedge (\neg(\neg z)))$

$\longrightarrow (x \wedge y) \vee ((\neg y) \wedge z)$

$\longrightarrow ((x \wedge y) \vee (\neg y)) \wedge ((x \wedge y) \vee z)$

$\vdots$

Substitution

$[x \mapsto e_1]\, e_2$ "Substitute e_1 for x everywhere inside e_2."

eg. $[n \mapsto 42]\,(n + n^2)$

$\longrightarrow 42 + 42^2$

eg. (any variable)

$e ::= n \mid e + e \mid \underline{\text{let}}\ x = e\ \underline{\text{in}}\ e$

$\text{let } n = 42 \text{ in } n + n$

$3 + (\text{let } x = 5 \text{ in } (\text{let } z = x + 2 \text{ in } z + x))$

New rewrite rule: (n: actual integer)

$\text{let } x = n \text{ in } e \longrightarrow [x \mapsto n]\, e$

$3 + (\text{let } x = 5 \text{ in } (\text{let } z = x + 2 \text{ in } z + x))$

$\rightarrow 3 + [x \mapsto 5](\text{let } z = x + 2 \text{ in } z + x)$

$\rightarrow 3 + (\text{let } z = 5 + 2 \text{ in } z + 5)$

$\rightarrow 3 + (\text{let } z = 7 \text{ in } z + 5)$

$\rightarrow 3 + (7 + 5)$

$\rightarrow 3 + 12$

$\rightarrow 15$