ALGEBRA AND DATA TYPES: CHEAT SHEET

JUSTIN POMBRIO

Type	Expr
`!`	0
`()`	1
`bool`	2
`u8`	256

Type	Expr
`(x, y)`	$x * y$
`Result<x, y>`	$x + y$
`[x; n]`	x^n
`fn(x) -> y`	y^x

Type	Expr
`String`	`String`
`Option<x>`	$1 + x$
`HashSet<x>`	2^x
`Vec<x>`	$\frac{1}{1-x}$

Algebraic Law	Equation	Refactoring	Data Type Example
Commutativity	$x * y = y * x$	Reordering	`(x, y)` $\equiv$ `(y, x)`
Associativity	$(x * y) * z = x * (y * z) = x * y * z$	Rearranging	`((x, y), z)` $\equiv$ `(x, (y, z))` $\equiv$ `(x, y, z)`
Identity	$1 * x = x$	Unit Elimination	`((), x)` $\equiv$ `x`
Absorbtion	$0 * x = 0$	Never Contagion	`(!, x)` $\equiv$ `!`
Commutativity	$x + y = y + x$	Reordering	`Result<x, y>` $\equiv$ `Result<y, x>`
Associativity	$(x + y) + z = x + (y + z) = x + y + z$	Rearranging	(can rearrange variants within enums)
Identity	$0 + x = x$	Never Elimination	`Result<x, !>` $\equiv$ `x`
Distributivity	$x * (y + z) = x * y + x * z$	Lowering Data	`(x, Result<y, z>)` $\equiv$ `(Result<x, y>, Result<x, z>)`
Defn. of Exp.	$x^n = x * x * ... * x$ (n times)	Array $\leftrightarrow$ Tuple	`[x; 3]` $\equiv$ `(x, x, x)`
Identity	$x^1 = x$	Short Array	`[x; 1]` $\equiv$ `x`
Exp. Law 0	$1^x = 1$	Boring Array	`[(); 1]` $\equiv$ `()`
Exp. Law 1	$(x^n)^m = x^{(n*m)}$	Array Flattening	`[[x; n]; m]` $\equiv$ `[x; n*m]`
Exp. Law 2	$x^{(n+m)} = x^n * x^m$	Array Splitting	`[x; n+m]` $\equiv$ `([x; n], [x; m])`
Exp. Law 3	$(x * y)^n = x^n * y^n$	AoS $\leftrightarrow$ SoA	`[(x, y); n]` $\equiv$ `([x; n], [y; n])`
Equiv. Reprs.	$x^n = x^n$	Func $\leftrightarrow$ Array	`fn(u8) -> x` $\equiv$ `[x; 256]`
Identity	$x^1 = x$	Thunk	`fn(()) -> x` $\equiv$ `x`
Exp. Law 0	$1^x = 1$	Void Func	`fn(x) -> ()` $\equiv$ `()`
Exp. Law 1	$(z^y)^x = z^{(x*y)}$	Currying	`fn(x) -> fn(y) -> z` $\equiv$ `fn(x, y) -> z`
Exp. Law 2	$z^{(x+y)} = z^x * z^y$	Func Splitting	`fn(Result<x, y>) -> z` $\equiv$ `(fn(x) -> z, fn(y) -> z)`
Exp. Law 3	$(y * z)^x = y^x * z^x$	Func Splitting	`fn(x) -> (y, z)` $\equiv$ `(fn(x) -> y, fn(x) -> z)`

The surpringly close correspondence between algebra and data type refactorings. Examples are given in Rust, but the refactorings are language independent.

(Thanks to Florian Strobl for pointing out a couple of typos.)