ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Associative Operations

Parallel Programming in Scala

Viktor Kuncak

Associative operation

Operation `f: (A,A) => A` is **associative** iff for every x, y, z:

$$f(x, f(y, z)) = f(f(x, y), z)$$

Consequence:

- two expressions with same list of operands connected with $\otimes$, but different parentheses evaluate to the same result
- reduce on any tree with this list of operands gives the same result

Associative operation

Operation `f: (A,A) => A` is **associative** iff for every x, y, z:

$$f(x, f(y, z)) = f(f(x, y), z)$$

Consequence:

- two expressions with same list of operands connected with $\otimes$, but different parentheses evaluate to the same result
- reduce on any tree with this list of operands gives the same result

Which operations are associative?

A different property: commutativity

Operation `f: (A,A) => A` is **commutative** iff for every x, y:

$$f(x, y) = f(y, x)$$

There are operations that are associative but not commutative

There are operations that are commutative but not associative

For correctness of `reduce`, we need (just) associativity

Examples of operations that are both associative and commutative

Many operations from math:

- addition and multiplication of mathematical integers (`BigInt`) and of exact rational numbers (given as, e.g., pairs of `BigInts`)
- addition and multiplication modulo a positive integer (e.g. 2^{32}), including the usual arithmetic on 32-bit `Int` or 64-bit `Long` values
- union, intersection, and symmetric difference of sets
- union of bags (multisets) that preserves duplicate elements
- boolean operations $\&\&, ||$, exclusive or
- addition and multiplication of polynomials
- addition of vectors
- addition of matrices of fixed dimension

Using sum: array norm

Our array norm example computes first:

$$\sum_{i=s}^{t-1} \lfloor |a_i|^p \rfloor$$

Which combination of operations does sum of powers correspond to?

Using sum: array norm

Our array norm example computes first:

$$\sum_{i=s}^{t-1} \lfloor |a_i|^p \rfloor$$

Which combination of operations does sum of powers correspond to?

```
reduce(map(a, power(abs(_), p)), _ + _)
```

Here $+$ is the associative operation of `reduce`

`map` can be combined with `reduce` to avoid intermediate collections

Examples of operations that are associative but not commutative

These examples illustrate that associativity does not imply commutativity:

- concatenation (append) of lists: `(x ++ y) ++ z == x ++ (y ++ z)`
- concatenation of `Strings` (which can be viewed as lists of `Char`)
- matrix multiplication AB for matrices A and B of compatible dimensions
- composition of relations $r \odot s = \{(a, c) \mid \exists b.(a, b) \in r \wedge (b, c) \in s\}$
- composition of functions $(f \circ g)(x) = f(g(x))$

Because they are associative, `reduce` still gives the same result.

Many operations are commutative but not associative

This function is also commutative:

$$f(x, y) = x^2 + y^2$$

Indeed $f(x, y) = x^2 + y^2 = y^2 + x^2 = f(y, x)$ But

$$\begin{aligned} f(f(x, y), z) &= (x^2 + y^2)^2 + z^2 \\ f(x, f(y, z)) &= x^2 + (y^2 + z^2)^2 \end{aligned}$$

These are polynomials of different growth rates with respect to different variables and are easily seen to be different for many x, y, z.

Proving commutativity alone does not prove associativity and does not guarantee that the result of `reduce` is the same as e.g. `reduceLeft` and `reduceRight`.

Associativity is not preserved by mapping

In general, if $f(x, y)$ is commutative and $h_1(z), h_2(z)$ are arbitrary functions, then any function defined by

$$g(x, y) = h_2(f(h_1(x), h_1(y)))$$

is equal to $h_2(f(h_1(y), h_2(x))) = g(y, x)$, so it is commutative, but it often loses associativity even if f was associative to start with.

Previous example was an instance of this for $h_1(x) = h_2(x) = x^2$.

When combining and optimizing `reduce` and `map` invocations, we need to be careful that operations given to `reduce` remain associative.

Floating point addition is commutative but not associative

```
scala> val e = 1e-200
e: Double = 1.0E-200
scala> val x = 1e200
x: Double = 1.0E200
scala> val mx = -x
mx: Double = -1.0E200

scala> (x + mx) + e
res2: Double = 1.0E-200
scala> x + (mx + e)
res3: Double = 0.0
scala> (x + mx) + e == x + (mx + e)
res4: Boolean = false
```

Floating point multiplication is commutative but not associative

```
scala> val e = 1e-200
e: Double = 1.0E-200

scala> val x = 1e200
x: Double = 1.0E200

scala> (e*x)*x
res0: Double = 1.0E200

scala> e*(x*x)
res1: Double = Infinity

scala> (e*x)*x == e*(x*x)
res2: Boolean = false
```

Making an operation commutative is easy

Suppose we have a binary operation `g` and a strict total ordering `less` (e.g. lexicographical ordering of bit representations).

Then this operation is commutative:

```
def f(x: A, y: A) = if (less(y,x)) g(y,x) else g(x,y)
```

Indeed `f(x,y)==f(y,x)` because:

- if `x==y` then both sides equal `g(x,x)`
- if `less(y,x)` then left sides is `g(y,x)` and it is not `less(x,y)` so right side is also `g(y,x)`
- if `less(x,y)` then it is not `less(y,x)` so left sides is `g(x,y)` and right side is also `g(x,y)`

We know of no such efficient trick for associativity