Deciding associativity in L0

Topics in Programming Languages
Datalogisk institut, Copenhagen University (DIKU)

Kristoffer Søholm & Oleksandr Shturmov
{soeholm,oleks}@diku.dk

May 12, 2013.

Abstract

In the context of parallelizable second-order array combinators, it is often necessary for the function argument to be an associative function. We describe a partial decider for the associative property of functions in the language L0, for implementation in the parallelizing compiler l0c, develped under the HIPERFIT project. The problem is undecidable in general. This paper explores the practical feasibility of such a decider in the context of function arguments to second-order array combinators.

Keywords: Assosiativity, parallelism, second-order array combinators, L0, l0c, HIPERFIT.

1 Introduction

An infix operator $\oplus : S \times S \to S$ is associative iff.

$$\forall\, x, y, z \in S\ .\ (x \oplus y) \oplus z = x \oplus (y \oplus z)\ .$$

One of the benefits of such functions is that a sequence of values $(x_k)_{k=1}^{n}$, $x_k \in S$, can be aggregated into a single value $x \in S$, using $n-1$ applications of $\oplus$, i.e. $x = x_1 \oplus \cdots \oplus x_{n-1} \oplus x_n$. This is more commonly known as a *reduction*. For instance, we can compute the sum of n integers using $n-1$ binary additions.

A straight-forward reduction strategy is $(\cdots((x_1 \oplus x_2) \oplus x_3) \cdots \oplus x_{n-1}) \oplus x_n$, or equivalently, $x_1 \oplus (x_2 \oplus \cdots (x_{n-2} \oplus (x_{n-1}, x_n)) \cdots)$. This is known as a linear left and right *fold*, respectively.

Reductions get more interesting on single-instruction multiple data (SIMD) architectures. A reduction on a SIMD machine can be performed in parallel time $O(\log n)$ on n processors, or $n/m + O(\log m)$ on m processors, using a technique called *parallel prefix scan*, or simply scan.

Scan is an interesting technique in and of itself as it can serve as a basic building block in the design of parallel algorithms, replacing primitive parallel memory referencing. If applicable, this technique typically reduces the asymptotic bounds by a factor of $O(\log n)$ [Blelloch].

Scan takes an infix operator $\oplus : S \times S \to S$, an identity value $i \in S$, such that $i \oplus i = i$, and a sequence $(x_k)_{k=1}^{n}$, $x_k \in S$. It produces the sequence $(i, x_1, x_1 \oplus x_2, \ldots, x_1 \oplus \cdots \oplus x_{n-1} \oplus x_n)$.

Parallel prefix scan performs a tree-like reduction rather than a linear reduction. The sequence $(x_i \oplus x_{i+1})_{i=1}^{n-1}$ is computed first, by pairwise applying $\oplus$ to the elements of the sequence. The sequence $((x_i \oplus x_{i+1}) \oplus (x_{i+2} \oplus x_{i+3}))_{i=1}^{n-3}$ is computed second, by pairwise applying $\oplus$ to the elements of the previous sequence, and so on until a scalar is reached.

It is therefore important for the correctness of the method that the function supplied to scan is an associative function.

Some compilers will assume that the user has supplied an associative function. Others will require for the function to be explicitly marked as "associative". Others still will try to deduce the associativity property, and warn the user if the function isn't clearly associative.

1.1 Undecidability

Undecidability of the property is proven by reduction from the halting problem. We construct a Turing machine (TM) M that takes as input a pair of values $x, y \in S$.

Assume there exists a TM A, that given an arbitrary TM, accepts iff M accepts or rejects on any input, and rejects otherwise. That is, A is a decider for the associativity problem.

We construct M such that it returns a constant $z \in S$ iff $A(M)$ accepts and loops otherwise. More formally:

$$M(x, y) = \begin{cases} z & \text{if } A(M) \\ M(x, y) & \text{otherwise} \end{cases}$$

M is associative iff M halts. The halting problem is undecidable in general.

1.2 An associative type

For ease of analysis, we introduce the concept of an *associative type* — a function type that unifies with the type of any associative function. An associative function consumes at least two values of type t, and returns a value of type t.

We refer to functions that unify with the associative type, but are not themselves associative as *not associative*. We refer to all other functions as *associative*.

In particular, we refer to functions that do not unify with an associative type as associative. This is to allow us to say that e.g. unary functions are associative. Although this does not make sense mathematically, our reasoning is that functions that do not unify with an associative type cannot be chained into a function whos type unifies with an associative type.

2 Builtins

2.1 Chaining

We consider first whether simply chaining applications of the operator is associative. That is, for each binary operator $\oplus : t \times t \to t$, we consider whether $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ by the L0 specification. Unary operators are trivially associative under chaining. For binary operators with arguments of two different types, as well as ternary and n-ary operators, it depends. Later, we'll allow intermixing of operators.

2.1.1 Arithmetic operators

L0 has a range of binary arithmetic operators, namely `+`, `*`, `-`, `/`, `pow`, and `~`, where the last operator is a unary negation operator. The operators defined on arguments of type either `int` or `real` (L0 is strongly-typed), yielding a value of the same type.

As of now, there are no restrictions in the L0 standard as to how `real`s are to be represented, so we confine our attention to `int`s. For `int`s we assume a p-bit two's complement representation.

Our observations are summarized in Table 1.

Operator	Chaining is associative
+	yes
*	yes
-	no
%	no
pow	no
~	yes*

Table 1: Associativity of chaining arithmetic operators.

2.1.2 Relational operators

L0 has a couple binary relational operators, namely `=`, `<`, and `<=`, having the obvious semantics. The operators are defined on arguments both of type either `int` or `real`, yielding a value of type `bool`.

Relational operators are not chainable in L0. These operators are only interesting in combination with other operators.

2.1.3 Logical operators

L0 has a couple logical operators, namely `&&`, `||`, and `not`, where the last is a unary negation operator. The operators are defined on arguments of type `bool`, yielding a value of type `bool`.

Our observations are summarized in Table 2.

Operator	Chaining is associative
`&&`	yes
`||`	yes
`not`	yes[*]

Table 2: Associativity of chaining logical operators.

2.1.4 Bitwise operators

L0 has a range of bitwise operators, namely `^`, `&`, `|`, `>>`, and `<<`, having semantics as in C. The operators are defined on arguments of type `int`, yielding a value of type `int`.

Our observations are summarized in Table 4.

Operator	Chaining is associative
`^`	yes
`&`	yes
`|`	yes
`>>`	yes
`<<`	yes

Table 3: Associativity of chaining bitwise operators.

2.1.5 Other functions

L0 has a range of other builtins, e.g. various second-order array combinators. Of particular interest to us are the functions `concat`, `min`, and `max`. The other builtins do not meet the requirement of unifiability of their type with an associative type, or fall prey to the fact that functions are not first-class citizens in L0. These operators are trivially associative.

Our observations are summarized in Table 4.

Operator	Chaining is associative
`concat`	yes
`min`	yes
`max`	yes

Table 4: Associativity of chaining bitwise operators.

[*]All unary operators are associative by definition.

2.2 Combining

Light's associativity test [Kehayopulu & Argyris].

References

[Kehayopulu & Argyris] Niovi Kehayopulu and Philip Argyris. *An algorithm for Light's associativity test using Mathematica*. Journal of computing and information, 3(1): 87–98. ISSN 11803886.

[Blelloch] Guy Blelloch. *Scans as Primitive Parallel Operations*. 1987. IEEE Transactions on Computers. Volume 38. 1526–1538.