algo-lang Documentation

Michael Kareithi

Nov 03, 2018

Contents:

Warning: This documentation is still a work in progress, as the project itself is a work in progress.

Alp (**Al** gorithm **P** rocessor) is a simple language made for designing algorithms. It is purposefully simple in its grammar, as to encourage users to break down the design problem into its smallest chunks, and compose working computations out of those.

The documentation here covers the language itself, ways to write and evaluate algorithms in the language (with the repl and such), and the architecture of the Alp interpreter.

CHAPTER 1

The Alp Language

The language name is still a work in progress

1.1 Why use Alp ?

The purpose of the Alp language is to provide a very simple, yet expressive interface for designing algorithms. In theory, the features in the Alp language can be used to represent any and all possible computations. That is to say, it is 'Turing Complete'. The goal here is to enable design of algorithms, outside the context of any one programming language. Alp can therefore be viewed as an attempt to create a formal system for defining and testing algorithms, that can then be implemented in some programming language X. For that reason, it is not a language to be used to build programs with (though technically it can). It is a language made to be *read* more than it is *run*.

This document is meant to be read from start to finish. As there are some explanations of things encountered early on that are deferred to later sections.

1.2 Core language features

The core features of the language can be split into two subgroups. That is:

1. **Data representations**
2. **Data transformations**

That's it. Nothing else. The entirety of the Alp language is representing and performing operations on data. Again, the reason for this simplicitly is to make the language as abstract as possible. The closer it is to the minimal definition of a Turing Complete language, the simpler it is to understand and implement in any other language. The following sections cover these two aspects of the language, in order.

Note: Alp single line comments are denoted with `//` and mutliline comments are denoted with `//--` for opening and `--//` for closing

---

1.2.1 Data

Data representation in algol can be done in three ways:

1.2.1.1 Literals

The Alp language currently involves only one type of number. That is, Natural numbers (), which can be shown represented as literals:

```
1  // Natural
27 // Natural
0  // Natural
-1 // Not a valid literal
```

Note: More number sets will be added to the language specification in the future.

1.2.1.2 Variables

Objects in Alp are all mutable, and can be defined as shown:

```
SOME_VAR <- // some expression that yields a value
```

Note: The capital lettering for the name of the variable is not need. It is an encouraged convention however, as it helps distinguish between variables and other language constructs we will see later.

The name `SOME_VAR` is the name of our variable. Variable names must begin with either a letter, dash or underscore. All variables can be re-assigned to a new value with the same notation:

```
SOME_VAR <- // some expression
```

It is also possible for a variable to reference itself when being reassigned:

```
SOME_VAR <- SOME_VAR
```

The above assignment does nothing, clearly, but it serves to show the fact stated above. A somewhat more useful example can be shown as:

```
SOME_VAR <- SOME_VAR + 1
```

This would be equivalent to incrementing `SOME_VAR` by 1. Integer operations, like the addition operator, shown above are covered later in the language.core.control.operators section.

Note: At this point in time, the Alp Interpreter supports . Support for strings and other numeric types, however, is a priority and will be added very very soon.

Note: Support for other constructs such as recursion (explicitly) will also be added to the Alp Interpreter at a later date.

1.2.1.3 Arrays

Alp arrays are mutable int-indexed lists of that can contain an arbitrary number of variables. The syntax for indexing, or setting the value of an object in an array is as shown:

```
ARR[x] <- // some expression
```

The syntax when creating a new array is as follows:

```
ARR_NAME <- [1..n] // Creates a new empty array of size n
```

Where *n* is the length of the array to be created. Remember that algo arrays are 1-indexed, as readibilty is number one.

> ARR_1[3] <- 4 // ARR_1 did not previously exist. This creates it and assigns its element *x* the value 4.
> ARR_1[2] <- ARR_2[5] + 5 // ARR_2 does not exist. The program terminates with an error at this point.

Arrays can be passed around and operated on as a whole as variables are. Given that the operations and algorithms using them as input use them as arrays.

Variables and Arrays are are all that is needed to represent data in Alp. Next comes **Control Structures**.

1.2.2 Control

CHAPTER 2

Writing Algorithms in Alp

Now that you've learned all there is to know about the Alp language, time to write some algorithms!

CHAPTER 3

The Alp Interpreter

This document goes over the architecture of the Alp interpreter.