(ARG!)

Ryan Eagan, Mike Goldin, River Keefer, Shivangi Saxena

AKA: rse2119, mag2277, rdk2123, ss4733

September 30, 2015

Language overview

(ARG!) is a language that is responsive to programmer anger. By appending exclamation marks to statements the programmer can progressively reduce the languages type safety in assignment operations. By appending exclamation marks to the STOP command, the programmer can break out of nested function blocks. While (ARG!) is statically typed when used without exclamation marks, it includes the blah type (blah for whatever) which is able to store any type without coercion and allows functions to return multiple data types. (ARG!) tries to get out of the angry programmers way as much as possible, therefore all values are logically true except for the explicit boolean false. In idiomatic (ARG!) the programmer is encouraged to write functions which test presumptions and to assume that these tests will fail. (ARG!) supports the presumption of failure by allowing the programmer to specify functions which return a blah type. These functions return values are boolean false by default, and only set to a logically true value if the presumption is proven true in the course of function execution. This return value can then be interpreted as a value of any type using the ! operator.

What it's good for

Scientific computing

(ARG!) bears similarity to C in its ability to break type safety through casting, but permits greater granularity than C in how it sloughs off type safety. The blah type, furthermore, furnishes greater flexibility than C in allowing the use of types defined at runtime. For these reasons, (ARG!) may be an excellent choice for some scientific applications dealing with extremely large datasets. In instances where precision requirements may change over the length of a data pipeline, (ARG!) allows very cheap sloughing off of precision. Where types may not be known until they are computed, (ARG!) enables those data of those types to be selected at runtime, stored and interrogated to determine how they should be handled. (ARG!)s presumption of failure allows for the early termination of programs yielding unhelpful results which might otherwise run longer than necessary.

Shared CPU and Turing-complete blockchains

The STOP! feature and (ARG!)'s block structure enable granular composition of functionality on the basis of accrued computational cost. Amazons AWS, for instance, assigns users CPU credits which are spent per-computation and regenerate slowly over time. An (ARG!) compiler designed to compose CPU usage accounting functionality into programs could make use of the STOP! function to halt programs which expend all of their available credits. A Turing-complete blockchain like Ethereum could make similar use of STOP! to ensure programs do not spend more than a specified sum of the users cryptofuel on a given task, and abort if the limit is reached.

! and !!

Appended to the end of an assignment operation, ! and !! comprise (ARG!)s type-safety shutdown mechanism. A single ! will enable a cast from a smaller to a larger type, while a double !! will enable casting from larger to smaller types, where data will be lost rather than merely recontextualized.

STOP!

The STOP operator requires $n > 0$! operators be appended, each of which moves the program counter logically past the current code block. Because (ARG!) programs are themselves code blocks, STOP! can be used to exit.

The blah type

The blah type is large enough to store any other primitive (ARG!) type, and is therefore useful in allowing functions to return multiple types, or for a variable to be interpreted in multiple contexts based on some logic.

Everything is a block

All code between brackets {} comprises a block, and the program is implicitly wrapped in brackets.

Everything is true unless its false

All primitive types include a boolean bit which is set to true, except in the case of the boolean false.

Sample program

A program which checks if an ASCII character can be counted to within n increments of an integer, starting from 0.

```
#ACOMPILERCOMMAND
/* Logically-speaking, execution begins on the first line that doesnt begin with a comment
   or a '#', which blah reserve for compiler commands. */

/* Everything is a code block, including the program itself. When you STOP! out of a block
   you move to the line immediately following the block. When you STOP! out of the
   program block the same thing happens, but an exit() is always inserted by the compiler
   following the program block. Returns work like regular C returns. */

/* argv[] and argc[] are available as global variables. */
char countto = argv[1]
int attempts = argv[2]

/* blah is the "whatever" type. It can store any of the other primitive types. */
blah result = counttochar(countto, attempts)

/* Anything which is not a boolean false is true. All types have an extra boolean bit
   which is true by default. */
if(result == true) {
  /* Print result, a blah type, as a string. */
  PRINT("%s\n", result)
}

/* Exit by breaking out of the program block. */
STOP!

/* This function that uses integers to count up to a character. It can return any type
   using the blah type */
FUNCTION blah counttochar(char countto, int attempts) {

  /* Idiomatic (ARG!) presumes failure. blah assume that this function will "fail" by not
     reaching countto in feblahr than attempts incrementations. */
  blah result = false

  int i = 0
  WHILE(i < attempts) {

    /* Force i to be a character using '!'. */
    char test = i!
    IF(test == countto) {
      /* Set result, previously a bool stored in a blah, to a char stored in the blah. */
      result = countto
      /* Jump out of the if block and the while block. The use of two exclamation
         marks specified two levels of nest breakout-ness. */
      STOP!!
    }

    i++
  }

  return result
}
```