YAPL – Yet Another Programming Language

Functional Specification

By Stephen Monaghan

Project Overview:

YAPL is a programming language that is meant to be used to prevent catastrophic failure in both large scale systems, as well as small embedded systems. It is a deterministic programming language which means that it has a limited number of states that it can enter, and its path from each state can be easily determined. The reason for this is that if there is an issue with the program's execution, a clean version of the program can easily begin execution where the broken program left off, allowing for error avoidance, as well as un-interrupted execution. This will be accomplished by periodically sending reports to another server, updating it on pertinent information for the execution in case this server stops executing. This data will include current state, global variable values, and maybe a few more pieces of information depending on the program running.

The language will have a similar structure to C, but more constrained. It will have a specific syntax that must be followed, in the form of having “states” that each execute a small portion of the steps necessary to complete the overall goal of the program (be that a drone control program, or a communication system). It will also not include unlimited loops, or loops that will run an indeterminate number of times (while loops).

It will also have a simple means of communication with drones and cameras built in, allowing for the programmer to focus more on the issue to be solved with program, and less time focusing on interfacing with the drone, or the camera system in place. This will be accomplished in two ways. First, there will be a number of primitives, that apply to the drone, as well as the cameras for easy access to import data from those systems. It will also have a number of simple functions, that translate to more complex actions in the language used to communicate with the drones, or the cameras.

Lastly, this language will have support for straight C code execution. This means that if there is something that can be done more easily without breaking the deterministic nature of this language by using C, then the programmer can embed this code into the program. This however can create issues, with how the program runs, so that code will be run on a separate thread, so that if there is a problem with it takig too much time, or with it breaking the deterministic nature of the program, then execution of that thread can be canceled, sending the program into an error state.

Use Case:

The main goal of this language is to avoid catastrophic failures in important systems, where a catastrophic failure would cause widespread problems. One of these cases is drones.

In the case of drones, this language's purpose is two fold. The first being to be easy to use with the drone, as well as making failure less likely. The other is that in the case of a failure that the execution of the code is not interrupted, or at least is interrupted minimally.

The first goal is able to be accomplished by placing strict constraints on the programmer. By controlling what a programmer is able to do when issues that can arise with low level programming languages should be able to be avoided. The language, for instance will not allow the programmer to directly change memory on the computer. Also, the deterministic nature of the language will make it so that infinite loops (loops that do nothing to accomplish the programs task, and in fact delay the programs execution) will not occur, allowing for forward progress. The issue of error recovery is more nuanced.

Because this language is deterministic, the next step of the program can be predicted without having to execute the whole program over again. Because of this, if there is an error during execution, execution can be picked up somewhere else to finish the program. Take the case of a server failing during the execution of a drone program. First a second server that also has an exact copy of this program gets a signal telling it that there has been a server failure on the main copy of the program. That other server then is able to quickly load up the most recent copy of the data sent to it, and using that information begin code execution from where it believes the first server left off. All of this should happen almost instantaneously to prevent the drone from falling out of the sky, or from having to return to its initial location, thereby failing its mission.

Another case would be if there is some kind of error during the code execution, that would make it either begin to go down an incorrect path, or break all together (for instance a flipped bit). In this case, there would be one server with the code executing, as well as the back up, along with a “voting” program running in the background. The voting program runs three separate copies of the code, and checks to see if there is any discrepancies with how the program decides to progress. If an issue is detected, the voter decides which path to take, based on which path is predicted by the most copies of the program. That choice is then sent to the backup server to begin execution at that point (along with all the data from that correct copy of the program). Then while that backup server is running the program, the main server reboots the program, and then requests to take over execution from the back up server.

In the case of a hacking attempt compromising execution, the protocol would be similar to that of the error in execution. The difference being that the server that was compromised might be taken offline more permanently to figure out how the attack was performed, as well as to prevent any more breaches of the system.

Non Goals:

This language does not need to be able to solve every problem, or do everything. It just needs to do this one thing better than a general purpose language can. It does not need to have advanced memory management in the form of what C provides. The language will also not provide full coverage of all C based system calls, or C abilities.

It also does not need to provide absolute system security in the form of a virtual machine, in the way that Java does. The programmer must vet his program to be sure that it will not cause issues on the system, that is not part of this language. It is not meant to replace these languages in every case, just cases where failure needs to be fixed immediately to prevent more issues from arising.

The language will also not provide an interface with all drones, that would be almost impossible. Instead, it will focus on one kind of drone, and then be able to be expanded outwards to other drones that operate in a similar manner to it with some modifications by the programmer to suit their needs. And similarly it will not provide full support for all cameras, but will instead work with one type of camera, doing one type of image processing.

Style:

The style of this language will be very similar to C code. At the top will be the includes, of external files that are needed for other code execution, as well as any declarations of variables that will remain constant throughout the execution (like C defines). That will be followed by any global variable definitions. These definitions will be in the style of C. They will begin with a type, and any other applicable modifiers (such as arrays, or pointers), followed by the variable ID, and then if they are being set at the time, they can be followed by an equals sign, and then the value of the variable.

This is where the YAPL and C take different paths. In this language there will next be a section of global functions. These are functions that can then be called multiple times in different states. This is so that the programmer does not have to re write the same function multiple times in the program. These functions operate under the same constraints as the rest of the language, and cannot do anything to impede progress, or change the deterministic characteristics of the language.

Following the functions, there is a section for the states of the program. These states act like and are formatted like functions in C, however they do not have return types attached to them, because they do not return values. Instead, they just act as transitions between the code. Each state contains actions that it performs, in sequence. Then instead of returns, it has “transitions”. These transitions define where in the program/ finite state machine the program begins executing next. The program does not have to transition from one state to the state defined directly after it, instead, it can skip states below it, or jump back up to states defined above it, ones that have already been executed, or even itself. These transitions can be surrounded by if statements in order to determine where to jump to next in the machine.

