Cartesian Dual Products

I Have No Name and I Must Scream

Abstract

Cartesian products have long been used in programming. However, these are fundamentally limited, as they presuppose that the program is a deterministic and divisible entity, and hence cannot capture the power of the mind. Using categorical imperative theory, we construct its dual, the Cartesian dual product. We demonstrate how Cartesian dual products provide an alternative programming model to monads as introduced to programming by Gottfrod Leibniz. We also show how Cartesian dual products interact with other aspects of programming philosophy such as existentialist types and Objectivist-oriented programming.

1. Introduction

We expect the reader to already have been acquainted with the goals of this paper by this point. However, for the forgetful reader, we have provided the following introduction:

Cartesian products have long been used in programming. However, these are fundamentally limited, as they presuppose that the program is a deterministic and divisible entity, and hence cannot capture the power of the mind. Using categorical imperative theory, we construct its dual, the Cartesian dual product. We demonstrate how Cartesian dual products provide an alternative programming model to monads as introduced to programming by Gottfrod Leibniz. We also show how Cartesian dual products interact with other aspects of programming philosophy such as existentialist types and Objectivist-oriented programming.

2. Background

Categorical imperative theory [1], first introduced by Immanuel Kant in 1785, was first developed to lay a foundation for all metaphysics, and has more recently been applied as a foundation for mathematics and programming language theory. In categorical imperative theory, a program may only contain a statement if every other statement could be transformed to be the same thing. This gives us the theoretical power to create new programming constructs by old. One popular way to do so is to "reverse all the arrows," furthermore making the program run backwards.

DOI: http://dx.doi.org/10.1145/(to come)

3. Development

Consider the program below.

```
main :: Greatest X -> IO ()
main = return (fst (42, x))
```

In traditional contexts, this program would construct the Cartesian product $(42, x)$, and then take its first element, returning 42. However, because we use the Cartesian dual product instead, this program has an unseen mind which cannot be determined by observing its body. Thus, this program is actually a complete implementation of Pac-Man.

Note that the argument x is never explicitly passed to the program. This is because x is an ontological argument; that is, x is the greatest possible argument, and the greatest possible argument has the property of existing.

4. Applications

We implemented Conway's Game of Life in both a monadic and a dualist programming language. We simulated a stick-figure human in both, as depicted in Figure 4. In our monadic implementation, this simulation of a man disintegrated within 3 timesteps. In the dualist implementation, the simulation depicted the man walking. We speculate that this occurred due to divine intervention because bad things don't happen to good people.

5. Extensions

Cartesian dual products combine naturally with existentialist types, as the semantics of existentialist types is defined in terms of the self of the program, which Cartesian dual products naturally define.

Because of the highly discontinous behavior of programs with Cartesian dual products, the question has been raised of how to repeat behavior in such a program. We that this could be achieved by embedding the program within an eternal recurrence as defined by Nietzsche [2]. Similarly, we found Nietzsche's will to power a useful mechanism for enabling asynchronous behavior using Cartesian dual products.

We attempted to design a language integrating Cartesian dual products with Objectivist-oriented programming. After substantial difficulty, we determined this was impossible because Cartesian dual products violate the Non-Contradiction axiom assumed in Objectivist-oriented programming. We then tried to see if we could invoke an Objectivist-oriented program from a Cartesian-dual program, or vice versa. This experiment failed when the call to start a new process returned an error code, because an Objectivist program will never live for the sake of another program, nor allow another program to live for it.

6. Conclusion

As shown in our experiments, Cartesian duals offer a powerful mechanism for creating mindful programs with very little code. We

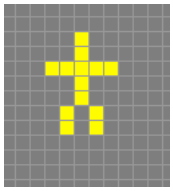

Figure 1. A simulated human

believe this represents a breakthrough in helping programs achieve consciousness.

References

[1] K. Immanuel. Groundwork of the metaphysics of morals. *Trans. Gregor Mary and Timmermann Jens. Cambridge, UK: Cambridge University Press*, 1785.

[2] F. W. Nietzsche and A. M. Ludovici. *Thus spake Zarathustra: a book for all and none*, volume 11. Macmillan, 1911.