Iamhowareyou

1 Introduction

Types, language, conditionals, information encoding, mathematics, and mathematical logic are ideas that hit at the heart of computer science. There's a lot here, and I plan to revisit these ideas in more depth later. This section will be another introduction of sorts – an intro to basic programming as such and an intro to computer science, the academic field. Let's start with types.

2 Types and Language

One of these programs is not like the other. I admit that I am new enough at Cedille that I did not know immediately how to replicate the examples in the other programs. For Cedille, it might be beneficial to build bottom-up and more slowly. For this section, we'll discuss types and languages in Cedille. Types are described more often by formal analogy than by any strict definition with types as propositions being the most famous example (1). According to the Stanford Encyclopedia of Philosophy, types originally just denoted distinctions between "individuals", "propositions", and "relations" where individuals and propositions have distinct base types and relations are built from connecting the base types together (2). These distinctions were necessary to avoid problems such as Russell's Paradox (2). A better explanation might come from types used in programs. Variables that hold data that's meant to be used as numbers are of type int or long or nat. Variables that hold data that's meant to be used as characters are often typed char. Data-structures such as lists are usually typed as well. Now about Cedille...

It's a programming language with one of the most interesting typing systems built today, called the Calculus of Dependent Lambda Eliminations. The types are extrinsic, they are erased when compiling to make its code faster running (less code to churn through when running). The types can be dependent (a type can depend on some code), they can be polymorphic (code can depend on types), they can be typeclasses (types can depend on types), they can also be dependent intersections (a type can be the intersection of two other types) (3). This got more complicated than I wanted. We'll come back to type theory. Let's look at language now.

One might say that data of type char or data of type int or data of type list or data of type string or data of type image are certain types of terms in

a formal language (it's not a very controversial statement). To the compiler or interpreter, your programming language is just strings (or tokens or what have you) until they become the physicists' problem at the hardware level.

We'll begin to make our own language within Cedille. Feel free to use whatever alphabet you want. I made a data-structure to hold the characters of English and a data-structure to combine characters into strings. The command line or terminal used for the other programs is accepting your typed input as strings and is also outputting data of type string. Strings are also used as the type that gets read in and out of compilers and assemblers that implement other programming languages. In some sense, strings are what both formal and natural language are usually typed as. We'll continue to build on both these data-structures and make better use of Cedille's unique type system in the future. For now, let's move onto interacting with the computer with strings.

3 If Then What

In the example programs except for Cedille, we have a narrative voice in the machine. Interacting with a computer in this way is what got me hooked on computer science. Making mini-text adventures for assignments felt like learning magic. We've already talked about your input being strings and the computer's output being strings too. Another important aspect that makes this magic possible is the conditional statement.

Conditional statements are extremely powerful. They allow more than one kind of outcome for a program. They add a level of complexity needed to make text-adventures, Turing complete programs, curry-isomorphic case-wise proofs, a whole bunch of things. By looking at the examples, one can see that they check whether some code is true or false during run time. When combined with either recursion, queues, or jumping back to a different part of memory, one gets a Turing Complete programming language. All ten of the languages in this tutorial are Turing Complete (this is rare for theorem provers like Cedille). Now onto the most pedantic and my favorite part of this section. Godel Encoded Numbers!

4 Godel Encoding and Information

The narrator in these code examples has a favorite number, a Godel encoded number specifically. Godel Encoding is an algorithm developed by Kurt Godel to uniquely compress a logico-mathematical statement into a number (4). Since proofs are a series of logico-mathematical statements connected by classical implication, one can uniquely encode proofs or propositions. Translating these ideas into numbers that can be added, multiplied, etc. allows for some brilliant things like meta-mathematics and meta-mathematical proofs (which themselves can be Godel encoded). It's very strange-loopy, and we'll continue to use and build on Godel numbering in this tutorial. This was meant mostly to be a

creative take on the conditional section of a Computer Science tutorial. There's a lot of stuff here, so I hope you check out the resources cited below.

5 Assignment

For an assignment, please browse the resources below for more in-depth background on these concepts. Then, make some programs that talk back to you. Whether it's a text adventure, a hello world program that asks for the time of day, a good morning program that asks you to affirm that it's a new day, or a spell caster that asks for an intent, make a program that uses these ideas and make sure it's something you want to make. These topics will hopefully only get more interesting, and we'll circle back to Godel because I'm a big fan. Thank you.

References

[1] F. Pfenning, "Lecture notes on the curry-howard isomorphism," 2004. https://www.cs.cmu.edu/ fp/courses/15312-f04/handouts/23-curryhoward.pdf.

[2] T. Coquand, "Type Theory," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, fall 2018 ed., 2018.

[3] https://cedille.github.io/.

[4] N. Wolchover, "How gödel's proof works," 2020. https://www.quantamagazine.org/how-godels-incompleteness-theorems-work-20200714/.