Typeful Programming: Moving Towards the Pragmatics of Programming

1 Typeful Programming

```
@article{cardelli1991typeful,
  title={Typeful Programming. Formal Descriptions of Programming Concepts},
  author={Cardelli, L},
  journal={Springer-Verlag, Berlin/New York},
  volume={45},
  pages={1989},
  year={1991}
}
```

Summary: Cardelli defines typeful programming and its use in a variety of different programming concepts and flavors. The paper argues that a sophisticated type system and the combination of static and dynamic type checking leads to programming that is free from certain classes of errors and software development that leads to robust and evolvable systems. He uses a toy language, Quest, to build up the principles of basic types in a language to complex typing constructs. He shows how these constructs can be applied to large and even huge software systems. Lastly, the paper explores how types can even be useful in unsound contexts, and how typing may still prove useful in categorizing where potential errors might occur.

Evaluation: The concepts covered in this paper had a large influence on the way certain programming languages are structured today. Many of the typing principles outlined in the paper are central to the design of languages such as Haskell and OCaml. Additionally, with gradual typing, typeful programming has become more prevalent even in languages that are commonly considered dynamically typed, such as JavaScript. For example, Typescript, a a language with the same syntax and semantics as JavaScript, brings the benefits of static types and typeful programming to the dynamic world of JavaScript programs. Overall, the paper makes a compelling argument for the importance of type systems in software development. However, it also predicts that languages like C will soon be considered unsuitable for creating large systems, and these predictions have not come to pass.