1 Points

- Criterion for config language design
- Exposition of designed language
- JSON examples and comparison with JSON Schema
- Future work: definitional equality, simple proof search, better JSON embedding, more examples

2 Story

Once apon a time, applications had configurations, but the configurations were messy and ill defined, having to choose between static languages with stronger guarantees, or computational languages with less guarantees. Enter our hero, Type Theory, who gives us a small but powerful language that lets us combine the static of power of JSON Schema with the computational power of Lua! With this new found power, order can be brought to configurations by statically defining the space of valid configurations for an application and then using the power of computation to define specific configurations in a concise way. And so, using this approach, applications and their configurations lived happily ever after.

3 Introduction

As programmers we have this vague notion of a configuration, the sort of knobs and dials that can be adjusted to change the behaviour of an application without actually changing the application itself. In this sense, the configuration for an application is a set of parameters, almost as if the application was a function with these parameters as its domain. A common problem with a configuration is that what exactly consists of a valid configuration is ill defined. Another problem which goes along with this is that the configuration is hidden inside the source code.

Try to backup these claims with evidence