Verifying a solution to a problem is significantly easier that finding one. This informal knowledge can be illustrated with a multitude of practical examples. Consider a Hamiltonian path problem aimed to determine whether a graph has a path in it that visits each vertex once. It requires little effort to check if a given sequence of vertices forms a path in the given graph. However, designing an algorithm to find such a path is considerably more challenging. There is a similar duality between type checking and type inference, program interpretation and program synthesis, and many other tasks.

This correspondence becomes apparent when considering the multi-modal nature of relational programming. In it, a single interpreter is capable of both verification and search based on which arguments are given as inputs.

[Interpreter] An *interpreter* for a language is a function `eval` which takes a program p written in the language and some input i and computes the output which corresponds to the semantics of the program p applied to the input:

$$\texttt{eval}(p, i) = p(i)$$

1 Generating Solvers from Verifiers

2 Relational Conversion

3 Issues