Equation Inference

by Sven Nilsen, 2017

In this paper I describe an algorithm that can infer all equations from data, up to some finite complexity decided by a filter set by the user. This can be used to find candidates for logical equivalence of functions by path transformation.

We have the following input:

x	y
false	false
false	true
true	false
true	true

Since \`x\` and \`y\` varies relative to each other, and this is the complete input for boolean binary operators, one can infer all equations that hold for all boolean binary operators. Likewise, if we have some data that is interesting for some other type of functions, e.g. \`concat : list → list\`, we can infer all equations that hold for all the input.

In general, there is an algorithm for inferring all equations that hold for some input data up to any finite complexity.

The algorithm starts with a collection of objects, called “the category”. The category contains:

1. Input, e.g. \`x : bool\` and \`y : bool\`
2. Constants, e.g. \`true\` and \`false\`
3. Single argument lambdas, e.g. \`add : \\(x) = \\(y) = x + y\`

New objects are added to the category by applying the single argument lambdas to other objects. For example, if \`add\` is applied to \`x\`, the returned lambda gets the name “add(x)”. If \`add(x)\` is then applied to \`y\`, the evaluated object gets the name “add(x)(y)”.

The filter contains list of names of length 1 or 2. If the list contains e.g. \`[“add(x)”]\` it means that no further exploration is desirable for applying \`add(x)\` to any other object. If the list contains e.g. \`[“add(x)”, “y”]\` it means that no further exploration is desirable for applying \`add(x)\` specifically to \`y\`.

A list with lists of length 2 is used to control exploration. If it contains \`[“add(x)”, “y”]\` then \`add(x)\` is applied to \`y\`. It is evaluated from top to bottom, such that new objects can be applied to other objects by adding a new list at the end.

Exploration is done before filtering, to not cancel out the objects that are already explored. This means that the same exploration choices and filter can be reused for all input data.

Any two objects in the category that have different names are checked for equivalence. If they are equivalent, they are added to a list of inferred equivalences.

An example algorithm is located under the Github project for path semantics in the file “dyon_experiments/category/src/category.dyon”