Generic Inference in Haskell

Alexander Knight

University of Queensland

Abstract

Problems requiring inference continue to crop up in many different fields of research, however the algorithms used to perform inference are individually rewritten in all these contexts. A Haskell library was created, providing methods for computing inference that operate on any values that satisfy the generic structure of a valuation algebra. While the implementation was successful and achieves the targeted computational complexities, it comes with performance overheads that may limit its usability.

Background

Inference is the process of taking a collection of separate pieces of information and then answering a query. We call each piece of information a valuation, and each valuation contains information about the values of certain variables. Our query is a set of variables, and the inference problem is to output a single valuation encapsulating all knowledgebase information about those variables. The challenge with solving an inference problem is that looking at any single valuation (piece of information) may not be sufficient to answer our query, forcing us to combine our valuations.

A database is a classic example of an inference problem. Each table represents a piece of information, and to answer our query we want all our query variables to be in one table. We can combine our tables using the natural join operation (matching up rows from each table if they share the same value in a common column).

To perform inference, a naive but valid approach would be to simply combine all our valuations. This produces one large valuation which we must be able to use to answer our query as it contains knowledge about every variable. In the context of databases, figure 2 displays an example of this process, using the example database of figure 1.

Table 1

Name	ID
Ben	1
Julia	3

Table 2

Noun	Colour
Sky	Blue
Grass	Green

Table 3

ID	Device ID
1	9
3	8

Table 4

Device ID	Device Type
9	Laptop
8	Phone

Figure 1. An example database of four tables.

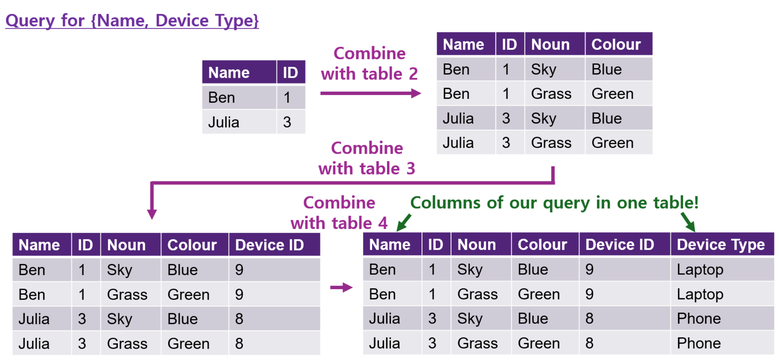

Figure 2. An example of solving an inference problem with poor efficiency.

The problem with this approach is that with every combination our valuations get larger and larger, making subsequent computations more and more expensive [1, p. xxv]. However, we can fix this issue if we may intersperse operations that reduce the size of our valuations. We call operations that reduce the size of our valuations **projections**. In our database example, this takes the form of dropping columns that we no longer require. Figure 3 shows the same process as in 2, but with projections interspersed.

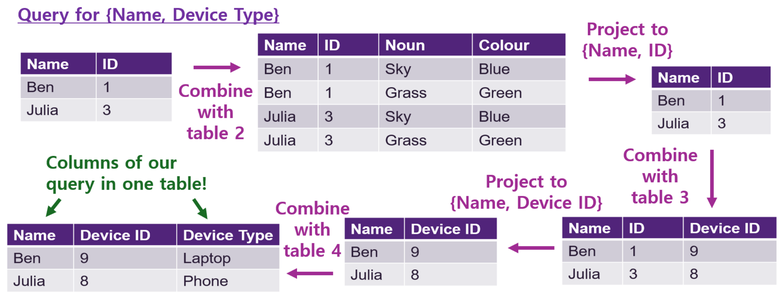

Figure 3. An example of solving an inference problem with efficiently.

The choice of when and where to perform combinations and projections is the core difficulty of performing inference. Many different inference algorithms exist that tackle this problem in different ways. Some algorithms provide better computational complexity for answering single queries, and others better complexity for answering a series of queries. An example of the former is **Fusion**, and of the latter, the **Shenoy-Shafer** algorithm.

Although we used the example of databases, we were able to describe the whole inference process just by using the terms **combination** and **projection**. The goal of this project was to write a library that performs inference for an arbitrary problem, provided it is told how to perform these combinations and projections.

Method

Several problems requiring inference and methods for computing generic inference were chosen from [1]. The chosen problems include determining the shortest path of a graph, performing Bayesian inference, and the implementation of the fast Fourier transform algorithm (FFT). The chosen generic inference methods were implemented in Haskell and then applied to solve the chosen problems. Benchmarks of each generic inference method on each problem were obtained.

Results (interface)

A library with the following interface was developed.

```
class ValuationFamily v where
  label   :: Var a => v a -> Domain a
  combine :: Var a => v a -> v a      -> v a
  project :: Var a => v a -> Domain a -> v a

type Domain a = Data.Set.Set a
type Var a = (Show a, Ord a)
type Valuation v a = (ValuationFamily v, Var a)

data Mode = BruteForce
          | Fusion
          | Shenoy { mode :: MessagePassingMode }
data MessagePassingMode = Threads | Distributed

queries :: (Valuation v a)
    => Mode -> [v a] -> [Domain a] -> IO [v a]
```

Results (validity & performance)

The generic inference algorithms are valid, outputting the expected values when applied to each problem. Their performance also tracks with the bounds on computational complexity described in [1, p. 66, p. 119, p. 371], lending evidence that they were implemented correctly. An example is illustrated in figures 4 and 5 which display run time increasing linearly with problem complexity. The **Threads** mode runs the computation over multiple threads, while the **Distributed** mode runs allows computation over multiple machines.

Notably, the benchmarks indicate that usage of the generic inference library comes with large overheads; figure 5 compares FFT performance against an existing Haskell library. Although the complexity class of both algorithms is the same, the difference in constant factors mean the generic inference implementation is orders of magnitude slower. Profiling reveals that performance is bottle necked by the collect and project operations. Future work may consider different join tree construction methods to reduce the number of collect and project operations invoked, or may investigate different implementations of the valuation family instances to improve the performance of each operation.

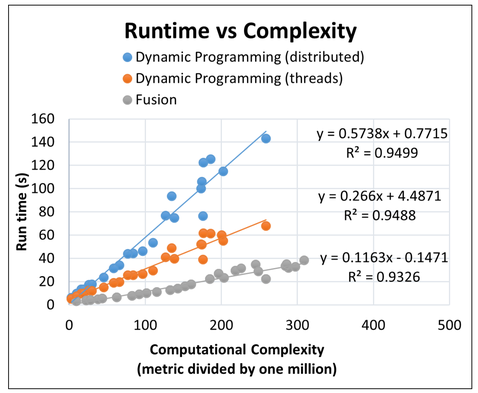

Figure 4. Runtime on shortest path problems of various complexities.

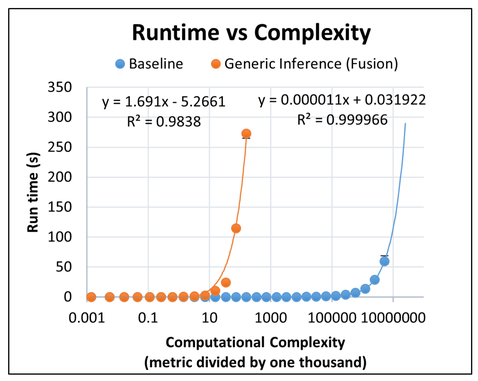

Figure 5. Runtime of FFT. Note trendlines are linear.

References

[1] M. Pouly and J. Kohlas, *Generic Inference: A Unifying Theory for Automated Reasoning*, en. John Wiley & Sons, Jan. 2012, Google-Books-ID: fbaUmmHzBqwC, ISBN: 978-1-118-01086-0.