Survey of Matrix Implementation in Functional Programming Languages*

Remi Desmartin[1], Grant Passmore[2], Ekaterina Kommendentskaya[1], and Matthew Daggit[1]

[1] Heriot-Watt University, Edinburgh, UK
[2] Imandra Inc. Austin TX, USA lncs@springer.com
http://www.springer.com/gp/computer-science/lncs
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

Abstract. The demand for formal verification tools for neural networks (NN) has increased as NNs have been deployed in a growing number of safety-critical applications. Matrices are a data structure essential to formalising NNs. Functional programming languages (FPLs), are particularly well-suited for automated reasoning tools which can be used for NN verification. In this paper, we survey different implementations of matrices in functional programming languages and how they impact the verification.

Keywords: Neural networks · Automated reasoning · Formal verification · Functional programming · Imandra.

1 Matrix as Lists of Lists

1.1 using Sized Lists or Vectors

Grant et al. ([1]) proposes 4 sparse list-based matrix implementations. They use an array-as-trees representation which allows to optimise for sparse arrays (subtrees where all the leafs are 0 are replaced by a 0-leaf).

Binary trees and lists of row-fragments: binary tree array of sparse Vectors defined as [(Int, [Double])]

A generalised envelope scheme: matrix is cut up in sections A quadtree scheme: Triangular matrix is split up in 2 triangular and a rectangular one. A standard quadtree structure is used for the rectangular matrix.

A Two-copy list of row-segments scheme: list of row-segments and list of column-segments in order to iterate over columns easily. Con: 2x more space is used; can be used to improve the 2 first previous methods (quadtree already bidimensional)

Pros of sparse list-based matrix representation: optimised for sparse matrices. Optimised for the specific operation considered in the paper (solving of linear systems of equations using a Cholesky scheme)

* Supported by organization x.

1.2 Refined Types

Coq/Mathcomp/SSReflex: the size of the matrix is defined as a refinement type and used to check that matrix have the appropriate size in multiplications Heras et al [2] discuss implementation; correctness is checked at compile-time.

This is also used in Starchild/Lazuli [3].

1.3 Arrays

Grant et al. [1] mentions using Haskell mutable arrays which are implemented using monadic operations. They stress that using a mutable array allows for modifying the array in place (thus saving memory), but it introduces "extra programming difficulties"; i.e. the use of monads makes the code less clear (as is the case in 1.4).

In our case, since IML is pure, there is no access to mutable arrays.

1.4 Monadic Operations to Check Size

Allows to check for valid matrix sizes when no refinement types are available. Intuitive first representation.

Drawback: introduces a lot of pattern matching, so a lot of split cases which increases the size of the program to check exponentially

Drawback: no optimisation for sparse matrices

2 Matrix as Functions

Imandra implementation; matrices are defined as total functions mapping indices to values. Theory of uninterpreted functions.

Woods [4] discusses the benefits of implementing matrices as functions in Agda.

3 Matrix as Maps

Imandra's functional programming language offers the possibility to use programming language data structures. Among these, it offers access to maps and records. Combining these two data structures allow us to implement matrices in a similar way to Matrices as functions 2 where we map coordinates to values:

$$m : \mathbb{N} \times \mathbb{N} \to \mathbb{R} \tag{1}$$

In order to encode the matrix's dimensions, we use OCaml's records, which allow us to bundle several values of different types together:

```
type 'a t = {
        rows: int;
        cols: int;
        vals: ((int*int), 'a) Map.t;
}
```

References

1. Grant, P.W., Sharp, J.A., Webster, M.F., Zhang, X.: Sparse matrix representations in a functional language. Journal of Functional Programming **6**(1), 143–170 (Jan 1996). https://doi.org/10.1017/S095679680000160X, https://www.cambridge.org/core/journals/journal-of-functional-programming/article/sparse-matrix-representations-in-a-functional-language/669431E9C12EDC16F02603D833FAC31B, publisher: Cambridge University Press
2. Heras, J., Poza, M., Dénès, M., Rideau, L.: Incidence Simplicial Matrices Formalized in Coq/SSReflect. In: Davenport, J.H., Farmer, W.M., Urban, J., Rabe, F. (eds.) Intelligent Computer Mathematics. pp. 30–44. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22673-1_3
3. Kokke, W., Komendantskaya, E., Kienitz, D., Atkey, R., Aspinall, D.: Neural Networks, Secure by Construction: An Exploration of Refinement Types. In: Programming Languages and Systems, vol. 12470, pp. 67–85. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-64437-6_4, series Title: Lecture Notes in Computer Science
4. Wood, J.: Vectors and Matrices in Agda (Aug 2019), https://personal.cis.strath.ac.uk/james.wood.100/blog/html/VecMat.html