Monad Sets –
Implementation and Mathematical Foundations

Ulrik Petersen

July 31, 2002

Contents

1 Introduction **1**

2 Monad Set Elements **2**

3 Monad Sets **2**
3.1 Introduction 2
3.2 Invariant 2

4 Operations on monad sets **3**

5 Relationships between MSEs **3**

A Proof of the uniqueness of monad sets **4**
A.1 Introduction 4
A.2 Proposition: 4
A.3 Proof: 4
A.3.1 Base case: 4
A.3.2 Induction step 6
A.3.3 Conclusion 7

1 Introduction

This document describes one efficient way of implementing arbitrary sets of monads. This way is used in the implementation of Emdros. This document is meant both as an introduction to the Emdros implementation of monad sets, and as a point of departure if you need to implement sets of monads yourself.

It is likely that you will need at least a basic implementation of sets of monads if you are using Emdros. If you are using C++, you can use the implementation provided in the EMdF library. If you are not using C++, first check to see whether there are any easy ways of implementing monad sets using some pre-existing library for your language, and that this easy way meets your requirements. If not, then you may consider implementing monad sets in the language you are using. If you do, please consider contributing your code to the Emdros code base so that others may benefit.

We start in section 2 by defining and motivating the "Monad Set Element" or "mse." This is a basic building block in the way of implementing sets of monads which is described in this document.

In section 3 we describe how to build sets of monads from mses.

In section 4 we describe a number of operations which it is desirable to have on monad sets. Most of these are implemented in the Emdros C++ implementation, so you can draw inspiration from there.

Then, in section 5 we describe six ways that MSEs can stand in a relationship to each other. These six ways turn out to be handy distinctions when implementing monad sets.

Finally, in an appendix, we give a formal proof that for any data-structure which obeys the invariants set down in sections 2 and 3, there is one and only one representation of each distinct set of monads, in other words, the representation is unique.

2 Monad Set Elements

A monad set element (or Monad_Set_Element or MSE or mse) is a pair of integers (or monads), called "first" and "last." The intended interpretation is that it denotes the range first..last. That is, it denotes a set of monads which includes every monad in the sequence of natural numbers starting at first and ending at last, with the two endpoints included.

The invariant is that first $\leq$ last. Thus last is never smaller than first, but may be equal to it. The latter means that the mse denotes a singleton set (i.e., a set with only one monad in it).

The first and last attributes of a Monad_Set_Element mse can be accessed as mse.first and mse.last.

3 Monad Sets

3.1 Introduction

From monad set elements, we build monad sets. We do this in such a way that the representation of any given set of monads is unique.

The monad set is seen as a list of monad set elements. The list is ordered. It may be a vector or a linked list. In the following, we describe the monad set as though the underlying data-structure were a vector. If many set-theoretic operations are to be performed on the set, then it is probably better to implement it as a linked list into which we can easily insert elements and take them out again. For space-efficiency, an array of monad set elements would probably be better.

There is an invariant on the list/vector/array of monad set elements. This invariant is responsible for most of the elegance of the implementation, as well as for the uniqueness property. We describe this invariant next.

3.2 Invariant

There is the following invariant on the Monad_Set_Element's stored in the vector:

For all Monad_Set_Elements mse in the vector, it is the case that:

1. Either
   (a) Its predecessor prev is empty (i.e., is not there), OR
   (b) prev.last+1 < mse.first,
2. AND either
   (a) Its successor succ is empty (i.e., is not there), OR
   (b) mse.last+1 < succ.first

This means:

$$\forall \text{ mse IN vector: } (\forall \text{ prev before mse: prev.last+1 < mse.first }) \wedge (\forall \text{ succ after mse: mse.last + 1 < succ.first })$$

Here the two nested for-all quantifiers take care of 1(a) and 2(a) by being vacuously true when prev and succ are not there.

This mathematical predicate captures these two intuitions:

1. All Monad_Set_Elements are *maximal*, in the sense that they extend as far as they can without violating the other intuition, which is that
2. The Monad_Set_Elements are *sorted* in such a way that, for any two Monad_Set_Element's A and B, where A is the direct predecessor of B,

$$A.last < B.first$$

which can be strengthened, by intuition 1., to:

$$A.last + 1 < B.first$$

which means that there is at least one monad in between each Monad_Set_Element.

4 Operations on monad sets

Here is a list of operations on monad sets which are desirable to have:

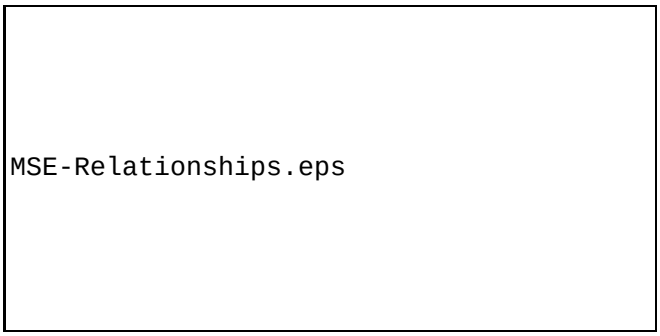
1. Set-theoretic union of two monad sets.
2. Set-theoretic difference between two monad sets.
3. Set-theoretic intersection of two monad sets.Traversal, getting each monad set element in turn.
4. Whether a monad set element is part_of another. The part_of relation is another name for the set inclusion operator, $\subseteq$.
5. Addition of MSE (which can be split into the two cases of add(first,last) and add(singleton)).
6. Set-theoretic equality (whether two sets of monads are identical).
7. Whether a monad is a member of a set of monads.
8. Whether a gap exists in a set of monads starting at a given monad.
9. What the next monad after a given monad is in a set of monads, if any. That is, either the next in the monad sequence or the first monad after a gap, or none if we were looking at the last monad of the set.

Most of these are implemented in the Emdros source code, so you can get inspiration from there. You may not need all of these, and 1-3 are available through the MQL language, so if performance is not of concern, you can use the MQL language rather than implementing these yourself. It stands to reason that processor-power will be faster in this case than calling up the MQL engine with a query and parsing the output.

5 Relationships between MSEs

The implementation of monad sets in Emdros takes advantage of the fact that any two MSEs can stand in one and only one of precisely six different relationships to each other. There are other conceivable relationships, but these six are what seem to be relevant. The six relationships are also orthogonal to each other, meaning that they are mutually exclusive. The six relationships are depicted in figure 1 on the following page.

You may want to use these distinctions, too, if you implement monad sets.



The two monad set elements, A and B, can stand in six different relationships to each other. The relationships are:

1. A.last < B.first
2. A.first < B.first AND A.last >= B.first AND A.last <= B.last
3. A.first >= B.first AND A.last <= B.last
4. A.first >= B.first AND A.first <= B.last AND A.last > B.last
5. A.first > B.last
6. A.first < B.first AND A.last > B.last

Figure 1: Ranges of monads and their interrelationships

A Proof of the uniqueness of monad sets

A.1 Introduction

We will give a detailed proof that two sets of monads are identical if and only if their representations are identical. The proof rests on the invariant.

Thus the representation of a given set is a unique representation – there is only one.

A.2 Proposition:

Any two Set_of_monad_ms represent the same set of monad_m's if and only if their monad_ms vectors are identical.

A.3 Proof:

Take two Set_of_monad_ms's A' and B'. Let A be an mse-vector representing A' and let B be an mse-vector representing B'.

"if": Assume that the two vectors A and B are identical. Then they trivially represent the same set.

"only if:" Assume that the two vectors represent the same set. We will now prove that the vectors are identical. The proof works by induction on the subscript operator i.

A.3.1 Base case:

i = 0.

To prove: A[0] = B[0].

Let mseA = A[0] and mseB = B[0].

Since the two sets are identical, certainly mseA.first == mseB.first, since this is the first monad of the sets. However, it is also the case that mseA.last == mseB.last. We will now see why.

There are two cases. Either there is only one mse in the vector A, or there is more than one.

Suppose there is only one mse in A. Then there is also only one mse in B, and mseA.last == mseB.last. To see this, consider the following: A' is not empty, since there is one mse mseA representing A'. Since the set of natural numbers is well-formed, and since A' is non-empty, A' has both a lower bound and an upper bound. The lower bound is mseA.first and the upper bound is mseA.last. If there is only one mse mseA in A, then A' is a single contiguous stretch of monads. Since A' and B' are identical, B' also consists of a single contiguous stretch of monads. By the invariant, the only way a single contiguous stretch of monads can be represented is with one mse. Otherwise, there would have to be two or more mse's representing the contiguous stretch of monads, but there is always at least one monad in between each mse. Therefore, the only way a single contiguous stretch of monads can be represented is with one mse. Since mseA.first == mseB.first, and since the two single contiguous stretches of monads are identical, it follows that mseA.last == mseB.last. Hence, if there is only one mse mseA in A, then there will be only one mseB in B, and mseA == mseB.

The other case is when there is more than one mse in A. Assume therefore that there is more than one mse in A. Then, since we have proved uniqueness when there is only one mse in A, there must also be more than one mse in B. In particular, A[0+1] = A[1] and B[0+1] = B[1] will exist.

Let mseA' = A[0+1] and mseB' = B[0+1].

Now, since the invariant holds, it is the case that

$$\text{mseA.last} + 1 < \text{mseA'.first}.$$

It is also the case that

$$\text{mseB.last} + 1 < \text{mseB'.first}.$$

Assume for the sake of contradiction that mseA.last $\neq$ mseB.last. Assume Without Loss of Generality that mseA.last > mseB.last. Then there must be at least one monad m in the set A' represented by A that is not in the set B' represented by B. (Note: This is not a renaming from the presentation of the proof above: It is the same A' and B'!) Next, we will see why.

There is at least one monad m = mseB.last+1 which is not in B but which is in A. To see this, note that, first, m cannot be in B', since

$$\text{mseB.last}+1 = m < \text{mseB'.first}.$$

Second, note that, since

$$\text{mseA.last} > \text{mseB.last},$$

m is in A', since the preceding inequality is equivalent to

$$\text{mseA.last} >= \text{mseB.last}+1$$

but

$$\text{mseB.last}+1 = m >= \text{mseB.first}$$

and

$$\text{mseB.first} = \text{mseA.first}$$

Thus m is in A', since it is between mseA.last and mseA.first. Since we have now proved that there is a monad m in A' which is not in B', we have a contradiction with our assumption that the two sets are identical.

Thus, there is a contradiction. Our assumption that mseA.last $\neq$ mseB.last must therefore be false. Therefore, mseA.last = mseB.last. Thus the base case holds.

A.3.2 Induction step

Assume that mseA' = A[i] and mseB' = B[i] are identical. Specifically, the following holds:

$$\text{mseA'.first} = \text{mseB'.first} \wedge \text{mseA'.last} = \text{mseB'.last}$$

We wish to prove that mseA = A[i+1] and mseB = B[i+1] are identical.
First, we prove that mseA.first = mseB.first. Second, we prove that mseA.last = mseB.last.
To see that mseA.first = mseB.first, assume for the sake of contradiction that

$$\text{mseA.first} \neq \text{mseB.last}.$$

Assume Without Loss of Generality that

$$\text{mseA.first} < \text{mseB.first}.$$

Then there must be at least one monad

$$\text{m} = \text{mseA.first}$$

which is in the set A' represented by A, but which is not in the set B' represented by B. To see this, note first that m is in A'. Second, note that m cannot be in B' by virtue of being in mseB. Note also that m cannot be in B' by virtue of being in mseB', because of the assumption that

$$\text{mseB'.last} = \text{mseA'.last}.$$

Since m is clearly not in B', m is in the set A' represented by A but not in the set B' represented by B. Since the two sets of monad_m's are assumed to be the same, we have a contradiction. Thus our assumption is false that mseA.first $\neq$ mseB.first, and thus it is true that

$$\text{mseA.first} = \text{mseB.first}.$$

Second, we prove that

$$\text{mseA.last} = \text{mseB.last}.$$

Assume, for the sake of contradiction, that

$$\text{mseA.last} \neq \text{mseB.last}.$$

Assume, Without Loss of Generality, that

$$\text{mseA.last} > \text{mseB.last}.$$

Then there is at least one monad, m = mseB.last + 1, which is in the set A' represented by A but which is not in the set B' represented by B.

To see this, note that if i+1 points to the last element in B, then certainly there is a contradiction, since then the last monad in the set represented by A is not in the set represented by B, since the vectors are sorted and mseA.last > mseB.last.

Assume, therefore, that mseB" = B[i+2] exists. Then, by the invariant,

$$\text{mseB.last} + 1 = m < \text{mseB''.first}.$$

Note that this means that m is not in the set represented by B, since it "falls between two chairs" or, rather, two Monad_set_elements.

Note also that m is in the set A' represented by A, since

$$\text{mseA.last} > \text{mseB.last}$$

which is equivalent to

$$\text{mseA.last} \geq m = \text{mseB.last} + 1 > \text{mseB.first} = \text{mseA.first}.$$

Thus, m is in the set A' represented by A, but not in the set B' represented by B, so the two sets are not identical, which we assumed. Therefore, there is a contradiction, and it is therefore false that $\text{mseA.last} \neq \text{mseB.last}$. Therefore, it is true that mseA.last = mseB.last.

We have thus proved the induction step.

A.3.3 Conclusion

We have thus proved that, if the sets represented by two vectors in two Set_of_monad_ms's are identical, then the vectors will be identical.

We have thus proved the proposition.