Capturing Matrix Algebras in Agda - An Experience Report

Michael Youssef
youssef@cs.rptu.de
RPTU Kaiserslautern-Landau
Kaiserslautern, Germany

ABSTRACT

Kleene algebras are algebras that provide operations for calculation that abstractly resemble composition, choice and iteration. Instances of the Kleene algebra where the underlying type is a matrix are sometimes referred to as Matrix Algebras. Since graphs can be represented using adjacency matrices, one can use Matrix Algebras to model graph algorithms such as reachability or shortest path among others. In this paper, a model of the Matrix Algebras is provided in Agda, a dependently typed functional programming language. Furthermore, options for modelling matrix data structures are considered and further development of graph algorithms other than the well established ones is discussed.

ACM Reference Format:
Michael Youssef. 2023. Capturing Matrix Algebras in Agda - An Experience Report. In *Proceedings of ACM Conference (IFL'23).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Kleene algebra over matrices provide a framework for manipulating graphs using the standard matrix operations of addition and multiplication. Such instances of the Kleene algebra are sometimes referred to as Matrix Algebras.

Working within this framework while potentially restrictive, is purely algebraic, and hence, does not suffer from the representation and manipulation issues of graphs data structures in a purely functional language. For non-pure languages, this is tackled using side effects manifested as mutable reference pointers.

Some of the work done to address graph representation and manipulation in pure languages includes Inductive Graphs presented by Erwig [5] and Algebraic Graphs by Mokhov [10]. These approaches fall somewhere within the spectrum of ad-hoc to fully algebraic.

In this paper, we will focus on an algebraic approach using Kleene algebras on sets of edges between nodes typically modeled using matrices. This means that the representation of the structure and the specification of the manipulation operations are irrelevant as long as the axioms of the algebra are respected.

Much work has been done in this regard already, most notably by Backhouse [1], Conway [3] and Kozen [8].

Kozen developed an axiom system for the Kleene algebra showing that square matrices do form a Kleene algebra given that the underlying elements of the matrix do themselves form a Kleene algebra with their corresponding operations. Conway and Backhouse studied the closure operation under matrices extensively and most of the work done on Kleene algebras on Matrices in general essentially traces back to their work.

IFL'23, August 2023, Braga, Portugal
2023. https://doi.org/10.1145/nnnnnnn.nnnnnnn

The driving tool of this paper is Agda, a dependently typed functional programming language. Using dependent types, we can jointly develop our data structures and algorithms, as well as prove properties about them.

Ultimately, we would like to discover which algorithms can be purposefully represented within the Matrix Algebra's framework, prove properties about said algorithms and derive more efficient instances of the inefficient specifications. This paper serves as a first step towards that goal.

2 KLEENE ALGEBRA OVERVIEW

There are many axiom systems and multiple interpretations of the Kleene algebra. For the sake of this paper, we use the axioms presented by Kozen [8].

As for the interpretations, when considering an algorithmic perspective, Kleene algebras are perhaps best described by Backhouse et al. [2] as an algebra for calculating using sequential composition, choice, and finite iteration.

In later sections, When considering the Matrix Algebras, the interpretation of automata on regular languages can be helpful for visual illustrations.

2.1 An Agda implementation

This section and the following ones contain fragments of Agda code. For the sake of brevity and visual clarity, some technical details are omitted. The full codebase can be found at [11].

We model the Kleene algebra as a record in Agda parameterized by the underlying carrier set as well as an equivalence relation. The operations are listed in Figure 1. Furthermore, we require that the operations preserve equality Figure 3. Moreover, the + operation defines a partial order on the elements of the carrier set as defined in Figure 2.

Figure 1: Operations

$$
\begin{aligned}
\_+\_ &: \mathit{Carrier} \to \mathit{Carrier} \to \mathit{Carrier} \\
\_\bullet\_ &: \mathit{Carrier} \to \mathit{Carrier} \to \mathit{Carrier} \\
\_* &: \mathit{Carrier} \to \mathit{Carrier} \\
\bot &: \mathit{Carrier} \\
I &: \mathit{Carrier}
\end{aligned}
$$

Figure 2: Partial Order

$$
\begin{aligned}
\_\leq\_ &: \mathit{Carrier} \to \mathit{Carrier} \to \mathit{Set} \\
a \leq b &= (a + b) \approx b
\end{aligned}
$$

Figure 3: Operations Preserve Equality

$$
\begin{aligned}
&+\text{-congruent} \;:\; a \approx c \to b \approx d \to a + b \approx c + d\\
&\bullet\text{-congruent} \;:\; a \approx c \to b \approx d \to a \bullet b \approx c \bullet d\\
&*\text{-congruent} \;:\; a \approx b \to a * \approx b *
\end{aligned}
$$

Figure 4: Closure Axioms

$$
\begin{aligned}
&\textit{left-unfold} \;:\; I + (a \bullet a *) \leq a *\\
&\textit{right-unfold} \;:\; I + (a * \bullet a) \leq a *\\
&\textit{left-induction} \;:\; a \bullet i \leq i \to a * \bullet i \leq i\\
&\textit{right-induction} \;:\; i \bullet a \leq i \to i \bullet a * \leq i
\end{aligned}
$$

Figure 5: Alternative Induction Axioms

$$
\begin{aligned}
&\textit{alt-left-induction} \;:\; b + a \bullet i \leq i \to a * \bullet b \leq i\\
&\textit{alt-right-induction} \;:\; b + i \bullet a \leq i \to b \bullet a * \leq i
\end{aligned}
$$

In Agda terms, the operations as well as the axioms are modeled as fields and the partial order $\leq$ is a declaration since it is defined in terms of the $\approx$ and $+$.

Additionally, $(\mathit{Carrier}\ , +\ , \bot)$ form an idempotent commutative Monoid, and $(\mathit{Carrier}\ , \bullet\ , I)$ form a Monoid. The two Monoids alongside axioms for distributivity and annihilation form a Semiring.

Moreover, the closure operator which can be defined as $A* = A^0 + A^1 + A^2 + \ldots..$ is described by the four axioms in Figure 4.

The induction axioms in Figure 5 are equivalent to their counterparts in Figure 4, and are usually quite useful for theorem proving.

Finally, we define two operations that represent a finite prefix of the closure operator in Figure 6.

Figure 6: Finite Repetition

$$
\begin{aligned}
A\,^{0} &= I\\
A\,^{1+n} &= A \bullet A\,^{n}\\
A\,*^{0} &= A\,^{0}\\
A\,*^{1+n} &= A\,*^{n} + A\,^{1+n}
\end{aligned}
$$

There is a plethora of useful theorems that we can derive from the axioms that are useful both in a general context and in the context of matrices, We list some of the relevant ones for this paper in Figure 7.

Figure 7: Kleene algebra Theorems

$$
\begin{aligned}
&\bullet\text{-monotonic-}\leq \;:\; a \leq c \to b \leq d \to (a \bullet b) \leq (c \bullet d)\\
&\textit{fin-induct} \;:\; I + (a \bullet a*^{n}) \leq a*^{n} \to a * \leq a *^{n}\\
&a^{n} \leq a*^{n} \;:\; a\,^{n} \leq a *^{n}
\end{aligned}
$$

3 THE MATRIX ALGEBRA

Kozen proved [8] that square matrices over a Kleene algebra with the usual operations of matrix addition and multiplication do form a Kleene algebra.

In this section, we reproduce the results in Agda using Conway's [3] formalization of the closure operator.

For our realization of a Matrix in Agda, we use a rather simplistic data structure restricted to $2^{n} \times 2^{n}$ matrices. show in Figure 8.

Figure 8: Block Matrix

$$
\begin{aligned}
&\textbf{\textit{data}}\ \mathit{BlockMatrix}\ (A\ :\ \mathit{Set})\ :\ \mathbb{N} \to \mathit{Set}\ \textbf{\textit{where}}\\
&\quad \mathit{Scalar}\ :\ A \to \mathit{BlockMatrix}\ A\ \mathit{zero}\\
&\quad \mathit{Block}\ :\ \{n\ :\ \mathbb{N}\}\\
&\qquad \to \mathit{BlockMatrix}\ A\ n\\
&\qquad \to \mathit{BlockMatrix}\ A\ n\\
&\qquad \to \mathit{BlockMatrix}\ A\ n\\
&\qquad \to \mathit{BlockMatrix}\ A\ n\\
&\qquad\quad \to \mathit{BlockMatrix}\ A\ (\mathit{suc}\ n)
\end{aligned}
$$

Index n in the definition holds information about the size where $\mathit{BlockMatrix}\ n$ means that the matrix has size $2^{n} \times 2^{n}$.

This simple variant serves the purpose of simplifying the rather lengthy proofs of the axioms. Obviously, this construction is restrictive, in later sections we provide a mechanism for transferring the proofs we do to any other *similar* structure. This is particularly helpful since the data structure influences the difficulty of the proofs, and conversely, we would prefer to be able to use more expressive structures and potentially more efficient operations.

The operations for summation and multiplication are just matrix addition and multiplication along with their neutral elements of a zero matrix and diagonal identity matrices. The case for the *Scalar* is just scalar addition or multiplication using the underlying Kleene algebra operations.

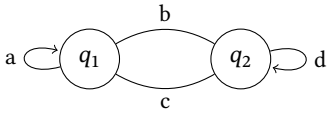
For the closure operator we use Conway's formula.

Figure 9: Conway's Formula

$$
\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{*} = \begin{bmatrix} (a + b \bullet d* \bullet c)* & a* \bullet b \bullet (d + c \bullet a* \bullet b)* \\ d* \bullet c \bullet (a + b \bullet d* \bullet c)* & (d + c \bullet a* \bullet b)* \end{bmatrix}
$$

The formula is perhaps better understood by using an automaton interpretation.

Figure 10: Automaton Representation



Each element in the matrix describes all the possible ways to get from a starting point to an ending point in an arbitrary number of transitions. For example, The (0 , 0) position describes all possible paths of arbitrary lengths starting from the node *q1* and ending back at *q1* itself.

3.1 Multiplication Fixed-Point

Cormen et al. [4, Section 23.1] proved that for the concrete instance of the shortest path problem, it takes $n - 1$ multiplication steps to arrive to a solution given that no edges carry negative weights, that is, for a matrix A, the shortest path solution is A^{n-1} and any further multiplication steps yield the same result $\forall\ m\ :\ A^{n-1} \approx A^{n+m}$.

The idea behind this is that at multiplication step k an additional multiplication step considers paths of length $k + 1$; therefore, doing $n - 1$ multiplications guarantees that the worst case path that passes through all the nodes has been considered.

We can generalize this result for Kleene algebras such that we have $A*^{n-1} \approx A*$ under the same conditions. We start by establishing the preconditions.

(1) The no negative weights condition generalizes to an inequality on the matrix where all elements are larger than *i*, the neutral element of multiplication of the underlying Kleene algebra as shown in Figure 11
(2) Another precondition we can assume but is not required is that the elements along the diagonal are exactly equal to *i*. This amounts to having an identity transition between a node and itself i.e. transitioning from a node to itself is a no-op. It is notable to mention that if this was not the case, we would have to do 2 $n - 1$ multiplication steps due to the need of covering every self loop for all n nodes as well. Furthermore, for the Kleene algebra, we would obtain the inequality $I \leq A$ this gives us a useful property of equality between multiplication and the finite prefix of the closure as shown in Figure 12.

Figure 11: Precondition

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \leq \begin{bmatrix} i & i \\ i & i \end{bmatrix}$$

Figure 12: Equality of Multiplication and Closure Prefix

$$\begin{array}{ll} & I \leq A \\ \Rightarrow & \{ \leq \textit{reflexive and} \bullet \textit{monotonic} \} \\ & A^n \bullet I \leq A^n \bullet A \\ \Rightarrow & \{ I \textit{ identity of } \bullet \} \\ & A^n \leq A^{1+n} \\ \Rightarrow & \{ \textit{by definition} \} \\ & A^0 \leq A^1 \leq A^2 \ldots A^n \leq A^{1+n} \\ \Rightarrow & \{ \textit{definition of} \leq \textit{and} *^n \} \\ & A^n \approx A*^n \end{array}$$

With everything in hand, we can proceed with the proof itself. We start of by taking the proof from Cormen et al. as a lemma and make the connection to the closure operation using the axioms of the Kleene algebra. In other words, we have to prove the lemma in Figure 13 for the full proof.

Figure 13: Equality of Closure and Finite Prefix

$$\textit{same-star} : \forall \{n : \mathbb{N}\} \{a : A\} \rightarrow a^{1+n} \approx a^{n} \rightarrow a* \approx a*^{n}$$

We start off with the *fin-induct* theorem in Figure 7 which is a direct result of the *alt-left-induction* axiom by substituting b for I and i for $a*^n$.

To obtain the first side of the inequality, we need a proof of the lemma $a^{1+n} \leq a^{n} \rightarrow I + (a \bullet a*^n) \leq a*^n$ but first, we derive the theorem $I + (a \bullet a*^n) \approx a*^{1+n}$ in Figure 14. We can then use the theorem to simplify the lemma to $a^{1+n} \leq a^{n} \rightarrow a*^{1+n} \leq a*^n$

Figure 16: Lemma

$$\begin{array}{ll} & a*^{1+n} \\ = & \{ \textit{by definition} \} \\ & a*^n + a^{1+n} \\ \leq & \{ \textit{transitivity of} \leq \textit{on } a^{1+n} \leq a^n \textit{ and } a^n \leq a*^n \} \\ & a*^n \end{array}$$

With the proof of the lemma $a^{1+n} \leq a^{n} \rightarrow a*^{1+n} \leq a*^n$ and the *fin-induct* theorem, we obtain one side of the inequality given the precondition, namely: $a^{1+n} \leq a^{n} \rightarrow a* \leq a*^n$. The other side of the inequality $a*^n \leq a*$ is fairly straight forward to prove and is left out of the paper but included in the associated repository. It is worth noting that this part of the proof is not specific to matrices. Now, if we combine the proof from Cormen et al. and the one presented here, we can make the connection between repeated multiplication that reaches a fixed point and the closure operator using the axioms.

Figure 14: Finite Left Unfold

$$
\begin{aligned}
& I + (a \bullet a *^{0}) \\
= \quad & \{ \textit{by definition} \} \\
& I + (a \bullet I) \\
= \quad & \{ \textit{by definition} \} \\
& a *^{1}
\end{aligned}
$$

(a) $n = 0$

$$
\begin{aligned}
& I + (a \bullet a *^{1+n}) \\
= \quad & \{ \textit{by definition} \} \\
& I + (a \bullet (a *^{n} + a\ ^{1+n})) \\
= \quad & \{ \bullet \textit{ distributes over } + \} \\
& I + a \bullet a *^{n} + a\ ^{2+n} \\
= \quad & \{ \textit{by induction} \} \\
& a *^{1+n} + a\ ^{2+n} \\
= \quad & \{ \textit{by definition} \} \\
& a *^{2+n}
\end{aligned}
$$

(b) $n = suc\ n$

4 REPRESENTATION INDEPENDENCE

We have demonstrated that our simple structure suffices to model the Matrix Algebra. Ideally, we would like to use other similar structures without having to redo the tedious proofs of all sixteen axioms all over again and potentially any other proofs.

The idea here is that if two structures are isomorphic and the operations defined on both structures exhibit the same behavior, then if one structure alongside its operations satisfy the axioms, the corresponding structure and operations should satisfy the axioms as well.

Given $(A\ , \approx\ , +\ , \bot\ , \bullet\ , I\ , *)$ and $(B\ , \approx^{B}\ , +^{B}\ , \bot^{B}\ , \bullet^{B}\ , I^{B}\ , *^{B})$ where the first 7-tuple already forms a Kleene algebra, we need to prove that the second 7-tuple also forms a KleeneAlgebra. Apart from the usual requirements of an equivalence relation on $\approx^{B}$ and that $+^{B}$, $\bullet^{B}$ and $*^{B}$ preserve equality, we require proofs for the propositions in Figure 17.

Roughly speaking, the propositions require an isomorphism between A and B, and that the operations exhibit the same extensional behavior.

Figure 17: Correspondence Requirements

$$
\begin{aligned}
& A{\rightarrow}B && : A \rightarrow B \\
& B{\rightarrow}A && : B \rightarrow A \\
& \textit{iso-to} && : B{\rightarrow}A\ (A{\rightarrow}B\ a) \approx a \\
& \textit{iso-fro} && : A{\rightarrow}B\ (B{\rightarrow}A\ a) \approx^{B} a \\
& A{\rightarrow}B\text{-}\textit{prop}\text{-}\approx && : a \approx b \rightarrow (A{\rightarrow}B\ a) \approx^{B} (A{\rightarrow}B\ b) \\
& B{\rightarrow}A\text{-}\textit{prop}\text{-}\approx && : a \approx^{B} b \rightarrow (B{\rightarrow}A\ a) \approx (B{\rightarrow}A\ b) \\
& A{\rightarrow}B\text{-}\textit{prop}\text{-}\bot && : A{\rightarrow}B\ \bot \approx^{B} \bot^{B} \\
& A{\rightarrow}B\text{-}\textit{prop}\text{-}I && : A{\rightarrow}B\ I \approx^{B} I^{B} \\
& A{\rightarrow}B\text{-}\textit{prop}\text{-}+ && : A{\rightarrow}B\ (a + b) \approx^{B} (A{\rightarrow}B\ a +^{B} A{\rightarrow}B\ b) \\
& A{\rightarrow}B\text{-}\textit{prop}\text{-}\bullet && : A{\rightarrow}B\ (a \bullet b) \approx^{B} (A{\rightarrow}B\ a \bullet^{B} A{\rightarrow}B\ b) \\
& A{\rightarrow}B\text{-}\textit{prop}\text{-}* && : A{\rightarrow}B\ (a *) \approx^{B} (A{\rightarrow}B\ a) *^{B}
\end{aligned}
$$

With what we have, we can prove that any equality proofs in A hold in B as well. The proofs are not of semantic nature, but rather involve the structure of Kleene algebra expressions. To capture that, we implement a data structure for modelling the expressions shown in Figure 18 and functions for semantic interpretation in B and another in A shown in Figures 19 and 20 where the fact that the two types are isomorphic is utilized.

Figure 18: Kleene Algebra Expressions

$$
\begin{aligned}
& \textbf{\textit{data}}\ KExpr\ (Carrier\ :\ Set)\ :\ Set\ \textbf{\textit{where}} \\
& \quad atom\ :\ (a\ :\ Carrier) \rightarrow KExpr\ Carrier \\
& \quad I^{s}\ \ :\ KExpr\ Carrier \\
& \quad \bot^{s}\ :\ KExpr\ Carrier \\
& \quad \_ +^{s} \_\ :\ KExpr\ Carrier \rightarrow KExpr\ Carrier \rightarrow KExpr\ Carrier \\
& \quad \_ \bullet^{s} \_\ :\ KExpr\ Carrier \rightarrow KExpr\ Carrier \rightarrow KExpr\ Carrier \\
& \quad \_ *^{s}\ :\ KExpr\ Carrier \rightarrow KExpr\ Carrier
\end{aligned}
$$

Figure 19: Semantic Interpretation in B

$$
\begin{aligned}
& \| \_ \|^{B}\ :\ KExpr\ B \rightarrow B \\
& \| \ atom\ a\ \|^{B}\ =\ a \\
& \| \ I^{s}\ \ \|^{B}\ =\ I^{B} \\
& \| \ \bot^{s}\ \|^{B}\ =\ \bot^{B} \\
& \| \ a +^{s} b\ \|^{B}\ =\ \| \ a\ \|^{B} +^{B} \| \ b\ \|^{B} \\
& \| \ a \bullet^{s} b\ \|^{B}\ =\ \| \ a\ \|^{B} \bullet^{B} \| \ b\ \|^{B} \\
& \| \ a *^{s}\ \|^{B}\ =\ \| \ a\ \|^{B} *^{B}
\end{aligned}
$$

Now we can transfer equality proofs from A to B. The proofs work by simply propagating the functions $B{\rightarrow}A$ and $A{\rightarrow}B$ to the atoms of the formulas. Figure 21 shows this correspondence and Figure 22, shows a function *derive* which is used to transfer the proofs.

Figure 20: Semantic Interpretation in A

```
‖ _ ‖^A : KExpr B → A
‖ atom a ‖^A = (B⟶A a)
‖ I^s ‖^A = I
‖ ⊥^s ‖^A = ⊥
‖ a +^s b ‖^A = ‖ a ‖^A + ‖ b ‖^A
‖ a •^s b ‖^A = ‖ a ‖^A • ‖ b ‖^A
‖ a *^s ‖^A = ‖ a ‖^A *
```

Figure 21: Propagating *A⟶B*

```
A⟶B^A≈^B : ∀ {a} → A⟶B ‖ a ‖^A ≈^B ‖ a ‖^B
A⟶B^A≈^B {atom a} = iso-fro
...
A⟶B^A≈^B {a •^s b} =
  ≈-transitive
    A⟶B-prop-•
    (•^B-congruent (A⟶B^A≈^B {a}) (A⟶B^A≈^B {b}))
...
A⟶B^A≈^B {a *^s} =
  ≈-transitive
    A⟶B-prop-*
    (*^B-congruent (A⟶B^A≈^B {a}))
```

We can transfer the proofs of the axioms across representations. For example, transferring the *left-unfold* axiom proof works as follows:

```
derive
  {(I^s +^s (atom a •^s (atom a *^s))) +^s (atom a *^s)}
  {atom a *^s}
  lef − unfold
```

There is one small caveat, the induction axioms are in fact implications, so this doesn't immediately work with the type of *derive*. Nevertheless, by requiring *B⟶A-prop-≈* we can do the same exact thing in reverse, that is, transferring proofs in *B* to *A*. For the implication we would then transfer the proof of the lhs which is in B to A, use the proof of the axiom in *A* to get the rhs which is in *A* as well, and finally transfer it back to *B*.

It is certainly noteworthy to state that the idea of transferring proofs across similar structures is not new and is studied in other domains in depth under a much more general setting, most notably in Homotopy Type Theory, but for our purposes here we stick to the simplistic approach that just serves our goal of transferring proofs of the Kleene algebra.

4.1 Application: Sparse Matrices

For an application of the matrix algebra where the matrix is sufficiently large and sparse, we can use the identity as well as the annihilation properties of matrices for addition and multiplication

Figure 22: Deriving Equality Proofs

```
derive : ∀ {s₁ s₂ : KExpr B}
  → ‖ s₁ ‖^A ≈ ‖ s₂ ‖^A → ‖ s₁ ‖^B ≈^B ‖ s₂ ‖^B
derive {atom a} {b} B⟶A[a]≈‖b‖^A =
    a
  ≈⟨ ≈-symmetric iso-fro ⟩
    A⟶B (B⟶A a)
  ≈⟨ A⟶B-prop-≈ B⟶A[a]≈‖b‖^A ⟩
    A⟶B ‖ b ‖^A
  ≈⟨ A⟶B^A≈^B {b}        ⟩
    ‖ b ‖^B
  ∎
...
derive {s₁ +^s s₂} {b} ‖ s₁ ‖^A + ‖ s₂ ‖^A≈‖ b ‖^A =
    ‖ s₁ ‖^B +^B ‖ s₂ ‖^B
  ≈⟨ +^B-congruent
    (≈-symmetric (A⟶B^A≈^B {s₁}))
    (≈-symmetric (A⟶B^A≈^B {s₂}))
  ⟩
    (A⟶B ‖ s₁ ‖^A +^B A⟶B ‖ s₂ ‖^A)
  ≈⟨ ≈-symmetric A⟶B-prop-+ ⟩
    A⟶B (‖ s₁ ‖^A + ‖ s₂ ‖^A)
  ≈⟨ A⟶B-prop-≈ ‖ s₁ ‖^A + ‖ s₂ ‖^A≈‖ b ‖^A ⟩
    A⟶B ‖ b ‖^A
  ≈⟨ A⟶B^A≈^B {b}              ⟩
    ‖ b ‖^B
  ∎
...
derive {a *^s} {b} ‖ a ‖^A * ≈‖ b ‖^A =
    ‖ a ‖^B *^B
  ≈⟨ *^B-congruent (≈-symmetric (A⟶B^A≈^B {a})) ⟩
    (A⟶B ‖ a ‖^A *^B)
  ≈⟨ ≈-symmetric A⟶B-prop-*                    ⟩
    A⟶B (‖ a ‖^A *)
  ≈⟨ A⟶B-prop-≈ ‖ a ‖^A * ≈‖ b ‖^A ⟩
    A⟶B ‖ b ‖^A
  ≈⟨ A⟶B^A≈^B {b}                             ⟩
    ‖ b ‖^B
  ∎
```

to short circuit the evaluation of operations. The idea is that we can replace sub-matrices with representatives of these identities. To do that, we augment the block matrix implementation with two constructors that represent the identities as show in Figure 23.

If we now modify the implementations of the addition and multiplication to short circuit the evaluation whenever we have the representatives, we can again prove that we have a Matrix Algebra.

While this is doable, the fact that we now have a redundant datastructure will make it extremely tedious. This is due to the fact that the number of case splits on parameters of the *SparseBlockMatrix*

Figure 23: Sparse Block Matrix

```
data SparseBlockMatrix (A : Set) : ℕ → Set where
  BlockBot  : (n : ℕ) → SparseBlockMatrix A n
  BlockOne  : (n : ℕ) → SparseBlockMatrix A n
  Scalar    : A → SparseBlockMatrix A zero
  Block     : {n : ℕ}
            → SparseBlockMatrix A n
            → SparseBlockMatrix A n
            → SparseBlockMatrix A n
            → SparseBlockMatrix A n
              → SparseBlockMatrix A (suc n)
```

potentially scales with the number of parameters as well, while for the *BlockMatrix* we only had a maximum of two regardless of the number of parameters. We can alternatively just bare the burden of proving the requirements in Figure 17 once, and then transfer any equality proofs from the *BlockMatrix* to the *SparseBlockMatrix* including the proofs of the axioms of the Kleene algebra.

5 COMBINING MATRIX ALGEBRAS

The algorithms presented in literature as instances of Matrix Algebras are pretty much primitive instances that arise from instantiating the underlying Kleene algebra. What has not been covered is how to combine these primitive results. For example, if we define two auxiliary functions:

(1) The function $\dot{\times}$ that multiplies all elements in corresponding positions $M \approx A \dot{\times} B \leftrightarrow \forall_{m,n} . M_{m,n} \approx A_{m,n} \bullet B_{m,n}$.
(2) The transposition function on matrices A^T.

We can use these two to specify the strongly connected components problem as $SCP\ (A) \approx A * \dot{\times}\ (A^T) *$ where we interpret the instance of the Matrix Algebra here as the reachability one.

The specification states that $A_{m,n} \approx true$ iff node m is reachable from n and vice versa, which is the specification of a strongly connected pair of nodes.

This is a simplified instance of the algorithm presented by Fleischer et al. [6] which fundamentally relies on the idea of combining forwards and backwards reachability queries.

6 FUTURE WORK

6.1 $n \times n$ Matrices

While proofs that $n \times n$ matrices fulfill the Kleene algebra axioms exist in literature, they involve some additional work. It is probably worthwhile trying to transfer proofs from the $2^n \times 2^n$ simplistic representation instead. The idea is that we can always augment an $n \times n$ matrix with unreachable nodes from all others so that it matches in the $2^n \times 2^n$ representation.

6.2 Combining Matrix Algebras

It is not apparent which algorithms on graphs can or can not be purposefully modelled using Kleene algebras. Moreover, By using algebraic constructions, one would hope to abstract away from the details of the algorithms to be able to find out if different algorithms are perhaps related at an abstract level.

6.3 Optimization

The Kleene algebra itself is perhaps a bit too general for some algorithms. A good example would be the single-source shortest-path algorithm. The plain Matrix Algebra instance computes the all-pairs shortest-path for the matrix, which is not very efficient if we are just interested in a single source. Höfner and Möller [7] actually used an extension of Kleene algebra, Kleene algebra with Tests [9] to derive Dijkstra's algorithm which is a more efficient algorithm for the single-source instance. The advantage of adding Tests in this context is that it adds the notion of guards to the Kleene Algebra's calculation model. Once again, the question is whether this method can be applied to other algorithms modelled using the Kleene algebra as well? If not, are other extensions of the Kleene algebra useful in this context?

REFERENCES

[1] Roland Carl Backhouse. 1976. *Closure algorithms and the star-height problem of regular languages.* Ph. D. Dissertation. Imperial College London, UK. https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.448525
[2] Roland C. Backhouse, Dexter Kozen, and Bernhard Möller. 2002. *Applications of Kleene Algebra (Dagstuhl Seminar 01081).* Dagstuhl Seminar Report 298. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany. 1–21 pages. https://doi.org/10.4230/DagSemRep.298
[3] John H. Conway. 1971. Regular algebra and finite machines.
[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.
[5] Martin Erwig. 2001. Inductive Graphs and Functional Graph Algorithms. *Journal of Functional Programming* 11 (04 2001). https://doi.org/10.1017/S0956796801004075
[6] Lisa K. Fleischer, Bruce Hendrickson, and Ali Pınar. 2000. On Identifying Strongly Connected Components in Parallel. In *Parallel and Distributed Processing*, José Rolim (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 505–511.
[7] Peter Höfner and Bernhard Möller. 2012. Dijkstra, Floyd and Warshall Meet Kleene. *Form. Asp. Comput.* 24, 4–6 (jul 2012), 459–476. https://doi.org/10.1007/s00165-012-0245-4
[8] Dexter Kozen. 1991. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Information and Computation* 110, 214 – 225. https://doi.org/10.1109/LICS.1991.151646
[9] Dexter Kozen. 1997. Kleene Algebra with Tests. *ACM Trans. Program. Lang. Syst.* 19, 3 (may 1997), 427–443. https://doi.org/10.1145/256167.256195
[10] Andrey Mokhov. 2017. Algebraic Graphs with Class (Functional Pearl). In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell* (Oxford, UK) *(Haskell 2017)*. Association for Computing Machinery, New York, NY, USA, 2–13. https://doi.org/10.1145/3122955.3122956
[11] Michael Youssef. 2023. Matrix Algebras. https://pl-git.informatik.uni-kl.de/youssef/matrixalgebras