Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Calculational graph algorithmics: reconciling two approaches with dynamic algebra

K. Clenaghan

Computer Science/Department of Algorithmics and Architecture

CS-R9518 1995

Report CS-R9518
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Calculational Graph Algorithmics: Reconciling two Approaches with Dynamic Algebra

Kieran Clenaghan

Department of Computing Science
The University of Glasgow
Glasgow, Scotland

Abstract

One concern in making the calculation of algorithms a formal mathematical activity is succinctness of notation and proof. Here we consider two recent contributions to this concern that we believe to be valuable. The contributions show how matrix algebra and relation algebra, respectively, enhance succinctness in the formal calculation of algorithms for path problems (amongst others) in graphs. The contributions are independent, and use different notations and proof strategies. However, the differences can be reconciled. Here we show how to make this reconciliation. The reconciliation is valuable for two reasons. First, it provides a straightforward synthesis of overlapping aspects of the two independent pieces of work, revealing the common ground behind superficially different appearances. Second, it prompts consideration of a unifying abstract framework as the basis for further algorithm calculations. An appropriate unifying framework is shown to be *dynamic algebra*.

CR Subject Classification (1991): G.2.2, I.2.2.
Keywords & Phrases: program calculation, path problems, matrix algebra, relational algebra, dynamic algebra.
Note: This work was carried out while on leave during 1994 at CWI, Amsterdam. The author gratefully acknowledges the support of the CWI.

1. Introduction

The recent independent papers [Möl93] and [BEG94] illustrate succinct derivations of algorithms for path problems by algebraic calculation. [Möl93] uses a relation algebra based on formal language concepts, and [BEG94] uses matrix algebra. The seeming distinctiveness of the characterisations is superficial. The goal of this paper is to show, in a precise sense, how the two approaches relate. This is achieved by showing that Möller's calculations can be translated and carried out in the matrix algebra setting of [BEG94]. To do this, it is natural to ask what abstract algebraic framework unites the relation and matrix algebras which are used. It turns out to be the *dynamic algebra* of [Pra90].

We start by setting out the matrix algebra of [BEG94] as a dynamic algebra. Next, we choose one algorithm from [Möl93], the graph reachability algorithm, and we express its calculation using the matrix dynamic algebra. Then we compare this to the relation-algebra calculation of [Möl93], showing that Möller's calculations can be reinterpreted in the matrix dynamic algebra setting. A nice by-product is that one of Möller's proofs can be shortened by adapting the calculations of [BEG94]. The paper concludes with a few remarks on the potential for further work based on applications of dynamic algebra.

The reachability problem is just one simple algorithm that serves for comparison. Both [Möl93] and [BEG94] treat other algorithms. The main purpose of our paper is to make comparative remarks, not to pursue the effectiveness of algebraic calculation in algorithmics. However, we provide an appendix that perhaps does both. It advances our comparative study and, in so doing, illustrates more of the effectiveness of algebraic calculation. There we: (a) present a brief calculation of Dijkstra's single-

source shortest paths algorithm as a précis from [BEG94]; (b) we specialise it to the reachability problem; and (c) we outline its specialisation to Moore's shortest paths problem (that just counts the edges). The latter is significant because it is calculated from scratch in Möller's style in [MR93].

1.1 Background

The body of this paper is brief and sharply focussed. Here we comment on some background to provide additional motivation for the less well acquainted with algebraic algorithm calculation and the algebraic structures special to path problems in graphs.

Formal algorithm calculation concerns the specification of an object that we wish to compute, and the manipulation of this specification by mathematical laws into a form that is recognisably an algorithm. [Möl93] and [BEG94] choose some graph problems that admit succinct specification, and enjoy good support from established algebra. Much of their calculation is concerned with equational reasoning. This is in contrast to the more usual reasoning about graph algorithms, e.g. [Kin90], and should be more amenable to machine assistance. [Möl93] sticks primarily to recursive equations (i.e. functional programs), whilst [BEG94] uses imperative loop constructs with equational invariants. [Möl93] is concerned with showing how relation algebra is widely applicable and yields succinct formal developments. It does not dwell on the passage from recursive equations to imperative programs. [BEG94] is concerned with showing how algebraic terms and laws can guide the derivation steps. Like [Möl93] it is concerned with compactness of expressions and derivations, but unlike [Möl93], it tackles deeper stages of algorithm refinement, producing imperative program code.

The overlap of [Möl93] and [BEG94] is their treatment of path problems in graphs. The simplest overlap is the reachability problem that we use as a comparative vehicle in the paper. Path problems are a nice choice because of the generality of certain algorithms that exist [Car79], [GM84], and because the underlying pure algebra (semiring theory) has been enjoying recent active research [Gol92]. Much of the established literature on algebra applied to path problems has been in establishing generic algorithms, and cataloguing their instantiation to a diversity of problems. A generic algorithm typically solves a fixed-point equation based on a kind of semiring, and its instantiation is obtained by nominating a particular semiring. [Car79] and [GM84] contain extensive examples.

The work of [Möl93] and [BEG94] is distinctive in that it advances the formalisation of proofs of algorithms for path problems. A key feature is the algebraicisation of (finite) quantification by exploiting products with (finite) sets (or selector vectors), as illustrated in $S \cdot R$, where S is a set (or vector) and R is a relation (or matrix). This has a history in automata theory, as recently highlighted in that subject by Kozen [Koz94], and its abstract setting is the theory of semimodules [KS86]. It also has a history in relation algebra, in which sets may be disguised as special relations [SS93].

Both [Möl93] and [BEG94] are clear about their use of Kleene algebra (a special kind of semiring), but neither identifies an abstract semimodule algebra which governs their calculations. [Möl93] adds sets as a special kind of relation, but then has to qualify some laws to account for the two kinds of relation that may be involved. [BEG94] just uses the specific matrix algebra that one gets over an underlying Kleene algebra, and points out the special rôle of so-called *selector* vectors for representing sets. It is easy to see that dynamic algebra [Pra90] is a unifying semimodule algebra. We believe that this observation should, at least, be useful in encouraging a convergence of notation.

2. Matrix algebra as dynamic algebra

The basic calculational methods of [Möl93] and [BEG94] can be conveyed by considering the calculation of a simple algorithm for the graph reachability problem. We review this problem now in order to motivate our choice of matrix algebra. Throughout, we assume that graphs are finite (i.e. the node set is finite).

Definition 1 (Graph Reachability) Let $G = (V , E)$ be a graph where $E \subseteq V \times V$ is the edge relation between nodes. The graph reachability problem is to find, for a given set of nodes, $S \subseteq V$, all nodes reachable from any node in S by reflexivity and transitivity of E. In other words, it is the problem of finding the image of S through the reflexive transitive closure of E. Treating S as the constant relation $\mathbb{1} \times S$ the problem is succinctly captured as the computation of $S ; E^*$, where E^* is the reflexive transitive closure of E defined as usual by:

$$E^* \triangleq \bigcup_{n \in \mathbb{N}} E^n \text{ where } E^0 \triangleq I \text{ and } E^{n+1} \triangleq E^n ; E \tag{2}$$

Similarly, we can represent relations by Boolean matrices, and sets by Boolean vectors, in which case relational composition becomes matrix product, and taking the image of a set through a relation is vector-by-matrix product. The graph reachability problem can be rewritten as $S \cdot E^*$, where $\cdot$ is matrix product, and E^* is the sum of all positive powers of E. We shall adopt the matrix viewpoint because it easily generalises to the non-Boolean case which is important for many problems on edge-weighted graphs.

□

Definition 3 (Paths) Let $G = (V , E)$ be a graph. A path from node i_1 to node j is a sequence of nodes $i_1 i_2 ... i_n j$ such that there is an edge in E from node i_k to i_{k+1} for $1 \leq k \leq n-1$, and there is an edge from i_n to j. The nodes in a path preceding the last node are called the *antecedent* nodes (of the path). An A-path, for $A \subseteq V$, is a path whose set of antecedent nodes is contained in A. The A-path relation between nodes can be denoted by $(I_A \cdot E)^*$ where I_A is a partial identity matrix, introduced in the next section. We shall say that a node, j, is *A-reachable* from a node i if there is an A-path, from i to j. Similarly, an m-path is a path with m antecedent nodes.

□

2.1 A matrix algebra

In the interests of general applicability it is useful to develop matrix algorithms in terms of general (or abstract) matrix algebra. It is well-known that matrix algebra can be presented abstractly using module theory, or more generally, semimodule theory. Recall that a module, $\langle M , S , \bullet \rangle$, combines a group M and a ring S with a multiplication, $\bullet : M \times S \to M$ that satisfies natural distributivity, unit, and zero laws. Roughly, a semimodule is a module without subtraction and division, and as such M is required only to be a monoid, and S a semiring. The semimodule treatment is particularly useful in computing science, see for example [KS86].

Vector-matrix algebra is based on the semimodule $\langle S^V , S^{V \times V} , \cdot \rangle$ where V is an index set, S is a semiring, and multiplication is matrix multiplication. In interesting special cases, S is a semiring with a unary * operator that extends uniformly to matrices. For example, we may stipulate that S is countably complete [Heb90] and define * to be the sum of all positive powers (countable completeness means that addition extends to the summation of countable subsets, and associativity, commutativity, and distributivity generalise accordingly). Alternatively, we may stipulate that S is a Kleene algebra [Koz94] which is an idempotent semiring (addition is idempotent) with an axiomatisation of * that generalises countable completeness (in the case of idempotent semirings). There are interesting non-idempotent examples (e.g. counting paths), but idempotency is often satisfied, e.g. Boolean and min-max algebras [GM84].

A semimodule $\langle M , S , \bullet \rangle$ in which M is a Boolean algebra, and S is a Kleene algebra has been called a dynamic algebra in [Pra90]. If S is a Kleene algebra with $\{0 , 1\} \subseteq S$, then it is straightforward to

construct a dynamic algebra $\langle \{0\ ,\ 1\}^V\ ,\ S^{V\times V}\ ,\ \cdot \rangle$. This is particularly useful because $\{0\ ,\ 1\}^V \cong \mathsf{P}V$ and for a set $A \in \{0\ ,\ 1\}^V$, and an S-valued relation or matrix $E \in S^{V\times V}$, $A \cdot E$ denotes the image of A through E. This is exploited in [BEG94] for the calculation of Dijkstra's single-source shortest paths algorithm (which we recapitulate in the appendix).

For our purposes, we use just two special dynamic algebras, these are: $\langle \mathbb{B}^V\ ,\ \mathbb{B}^{V\times V}\ ,\ \cdot \rangle$ and $\langle I(\mathbb{B}^V)\ ,\ \mathbb{B}^{V\times V}\ ,\ \cdot \rangle$, where $I(\mathbb{B}^V)$ is the set of partial identity matrices (i.e. $I_A \in I(\mathbb{B}^V)$ is a matrix which is everywhere 0 except for a 1 in each position $(a\ ,\ a)$ for $a \in A$). We shall use the standard set notation for elements of $\mathbb{B}^V$ and matrix notation for elements of $I(\mathbb{B}^V)$. We illustrate this in completing the picture of I as a morphism $\mathbb{B}^V \to I(\mathbb{B}^V)$:

$$I_{A\cup B} \ = \ I_A + I_B \tag{4}$$

$$I_{A\cap B} \ = \ I_A \cdot I_B \tag{5}$$

$$I_\emptyset \ = \ 0 \ \textit{and} \ I_V \ = \ 1 \tag{6}$$

It is important to recognise the following relational interpretations.
Let $A\ ,\ B \in \mathbb{B}^V$, $R \in \mathbb{B}^{V\times V}$:

- $I_A \cdot R$ is the domain-restriction of R to A.
- $R \cdot I_B$ is the range-restriction of R to B.
- $A \cdot R$ is the image of A through R.

Here are some elementary properties that we shall use:

$$1 \ = \ I_V \ = \ I_{\overline{A}\cup A} \ = \ I_{\overline{A}} + I_A \quad \text{(identity decomposition)} \tag{7}$$

$$M \ = \ I_{\overline{A}} \cdot M + I_A \cdot M \quad \text{(matrix decomposition)} \tag{8}$$

$$B \cdot I_A \ = \ B \cap A \quad \text{(partial image)} \tag{9}$$

*2.2 The * operator*
For our example of graph reachability, we want the star operator on relations to be the transitive, reflexive closure, i.e. the sum of all positive powers (where multiplication is relational composition). The main work in deriving an algorithm from the specification $S \cdot E^*$ is the decomposition of the work performed by * so that we arrive at a recursive formulation in which * no longer applies (or becomes trivial).

We need not spell out the axiomatisation of Kleene algebra, nor the formal definition of countably complete semirings, but instead we just list the following properties of * which we use in our calculations (and which happen to be common to both kinds of algebra):

$$a^* \ = \ 1 + a \cdot a^* \ = \ 1 + a^* \cdot a \quad \text{(*-fixpoint)} \tag{10}$$

$$(a+b)^* \ = \ (a^* \cdot b)^* \cdot a^* \quad \text{(*-decomposition)} \tag{11}$$

$$a \cdot (b \cdot a)^* \ = \ (a \cdot b)^* \cdot a \quad \text{(*-leapfrog)} \tag{12}$$

Variants of the *-decomposition law are easily obtained by using *-leapfrog and commutativity of addition.

A useful further law derived from those above is the following which enables one to "unfold" occurrences of summands out of a starred sum. Let $c = a+b$.

(13) $$c^* = (1 + c^* \cdot b) \cdot a^* \quad \text{(*-unfold)}$$

The proof is a straightforward algebraic calculation. We give it here to illustrate our proof style which is borrowed from [BEG94]. Note, in particular, that an assumption used in a proof step is highlighted by a bullet.

$$\begin{array}{ll} & c^* \\ = & \quad \{ \quad \bullet \quad c = a+b \quad \} \\ & (a+b)^* \\ = & \quad \{ \quad \text{*-decomposition} \quad \} \\ & (a^* \cdot b)^* \cdot a^* \\ = & \quad \{ \quad \text{*-fixpoint} \quad \} \\ & (1 + (a^* \cdot b)^* \cdot a^* \cdot b) \cdot a^* \\ = & \quad \{ \quad \text{*-(de)composition} \quad \} \\ & (1 + (a+b)^* \cdot b) \cdot a^* \\ = & \quad \{ \quad \bullet \quad c = a+b \quad \} \\ & (1 + c^* \cdot b) \cdot a^* \end{array}$$

The following elementary propositions, specific to certain dynamic algebras, will also be used.

Proposition 14 (Guarded *) Let S be a semiring and consider the dynamic algebra $\langle I(\{0\,,\,1\}^V)\,,\, S^{V\times V}\,,\,\cdot\rangle$. If $A \cap B = \emptyset$ then $A \cdot (I_B \cdot M)^* = A$, where $A\,,\,B \in \{0\,,\,1\}^V$, $M \in S^{V\times V}$.

Proof

$$\begin{array}{ll} & A \cdot (I_B \cdot M)^* \\ = & \quad \{ \quad \text{*-recurrence, distributivity} \quad \} \\ & A + A \cdot I_B \cdot M \cdot (I_B \cdot M)^* \\ = & \quad \{ \quad A \cdot I_B = A \cap B = \emptyset \quad \} \\ & A \end{array}$$

□

Proposition 15 (*-Reflexivity) Let $A\,,\,B \in \mathbb{B}^V$ and $E \in \mathbb{B}^{V\times V}$ in the dynamic algebra $\langle I(\mathbb{B}^V)\,,\,\mathbb{B}^{V\times V}\,,\,\cdot\rangle$. If $A \subseteq B$ then $B \cdot E^* \cdot I_A = A$.

Proof

$$
\begin{array}{ll}
 & B \cdot E^* \cdot I_A \\
= & \quad \{ \quad \text{*-fixpoint and distributivity} \quad \} \\
 & B \cdot 1 \cdot I_A \cup B \cdot E \cdot E^* \cdot I_A \\
= & \quad \{ \quad A \subseteq B \quad \} \\
 & A \cup B \cdot E \cdot E^* \cdot I_A \\
= & \quad \{ \quad X \cdot I_A \subseteq A \quad \} \\
 & A
\end{array}
$$

□

We have borrowed a lot from the notation and calculations in [BEG94]. There are some minor differences, the main one being our use of I_A for the $\underline{A}$ of [BEG94] . We choose I_A because it is standard (I being a natural transformation), and because it strikes a good balance between the quality of the identity relation and its restriction to A. Our silent transfer between sets and their characteristic vectors is not present in [BEG94] but is consistent with the pursuit of conciseness expounded in that paper; so too is our use of context to distinguish vector-valued and matrix-valued variables.

3. Reachability: a matrix approach

There is an easy solution to the reachability problem (cf. definition 1). To compute $S \cdot E^*$ we take the nodes in S (since they are reachable by reflexivity), and we (recursively) solve the problem for the immediate successors of S in E . This is given by the algebra in just two steps. Define $r_B = B \cdot E^*$, (so that r_B represents the problem at some intermediate point, the whole problem being distinguished as r_S). We calculate:

$$
\begin{array}{ll}
 & r_B \\
= & \quad \{ \quad \text{definition of } r \quad \} \\
 & B \cdot E^* \\
= & \quad \{ \quad \text{*-fixpoint} \quad \} \\
 & B \cdot (1 + E \cdot E^*) \\
= & \quad \{ \quad \text{algebra} \quad \} \\
 & B \cup (B \cdot E) \cdot E^* \\
= & \quad \{ \quad \text{definition of } r \quad \} \\
 & B \cup r_{B \cdot E}
\end{array}
$$

This leads to the following recursive definition that will solve the reachability problem for acyclic graphs.

$$r_\emptyset = \emptyset \text{ and } r_B = B \cup r_{B \cdot E}$$

Termination is guaranteed because finiteness and acyclicity ensure that, for some n , there are no m-paths (i.e. $E^m = \emptyset$), for all $m > n$.

To obtain a definition that terminates for cyclic graphs, a little thought tells us that the recursive case can be restricted to a smaller graph, E restricted by removing edges leading from B . Edges

are used only for the nodes to which they lead, and once we have these the edges can be deleted. The deletion of edges leading from B is obtained by restricting the domain of E to nodes outside B. This is $I_{\overline{B}} \cdot E$. Thus we arrive at the following definition.

$$R(\emptyset\ ,\ E)\ =\ \emptyset\ \textit{and}\ R(B\ ,\ E)\ =\ B \cup R(B \cdot E\ ,\ I_{\overline{B}} \cdot E)$$

We need to show that this is correct and terminates. Correctness obliges us to show that R is consistent with the assumption $R(B\ ,\ E)\ =\ B \cdot E^*$. The base case is easy, and we are left to show the recursive case, i.e.:

$$B \cdot E^*\ =\ B \cup R(B \cdot E\ ,\ I_{\overline{B}} \cdot E)\ =\ B \cup (B \cdot E) \cdot (I_{\overline{B}} \cdot E)^*$$

This equation succinctly captures (a variant of) the breadth-first strategy for traversing graphs, so we shall give it the status of a lemma.

Lemma 16 (Breadth-first decomposition)

$$B \cdot E^*\ =\ B \cup (B \cdot E) \cdot (I_{\overline{B}} \cdot E)^*$$

Proof

$$\begin{array}{ll} & B \cdot E^* \\ = & \quad \{ \quad \text{matrix decomposition (8)} \quad \} \\ & B \cdot (I_{\overline{B}} \cdot E + I_B \cdot E)^* \\ = & \quad \{ \quad \text{*-unfold (13)} \quad \} \\ & B \cdot (1 + E^* \cdot I_B \cdot E) \cdot (I_{\overline{B}} \cdot E)^* \\ = & \quad \{ \quad \text{algebra} \quad \} \\ & (B \cup B \cdot E^* \cdot I_B \cdot E) \cdot (I_{\overline{B}} \cdot E)^* \\ = & \quad \{ \quad B \cdot E^* \cdot I_B\ =\ B \text{ (Prop. 15)} \quad \} \\ & (B \cup B \cdot E) \cdot (I_{\overline{B}} \cdot E)^* \\ = & \quad \{ \quad \text{algebra} \quad \} \\ & B \cdot (I_{\overline{B}} \cdot E)^* \cup B \cdot E \cdot (I_{\overline{B}} \cdot E)^* \\ = & \quad \{ \quad B \cdot (I_{\overline{B}} \cdot E)^*\ =\ B \text{ (Prop. 14)} \quad \} \\ & B \cup B \cdot E \cdot (I_{\overline{B}} \cdot E)^* \end{array}$$

□

Now we must justify termination: $I_{\overline{B}} \cdot E \subseteq E$ and if $I_{\overline{B}} \cdot E\ =\ E$ then $B \cdot E\ =\ \emptyset$, so either E is strictly reduced or we reach the termination case. Since E is finite there can be no infinite sequence of strict reductions, and $E\ =\ \emptyset$ leads directly to the terminating case.

A variation on the above solution is easily derived. Instead of successively removing edges leading from visited nodes, we can accumulate the visited nodes, as say, C. Then, $I_{\overline{C}} \cdot E$ (E domain-restricted to nodes outside C) is equal to E with all the edges leading from visited nodes deleted. Now, instead of taking the successors of B in (the reduced) E, we take the successors of B in $I_{\overline{C}} \cdot E$. This is captured by the following equations.

$$R'(\emptyset\ ,\ C)\ =\ \emptyset\ \mathit{and}\ R'(B\ ,\ C)\ =\ B \cup R'(B \cdot I_{\overline{C}} \cdot E\ ,\ B \cup C)$$

The intuition behind R' is that it computes the nodes which are $\overline{C}$-reachable from B. This is formalised as $B \cdot (I_{\overline{C}} \cdot E)^*$ (cf. Defn. 3). The whole reachability problem is the case $C = \emptyset$: $R'(S\ ,\ \emptyset)\ =\ S \cdot (I_{\overline{\emptyset}} \cdot E)^*\ =\ S \cdot E^*$. We now verify that R' is consistent with $R'(B\ ,\ C)\ =\ B \cdot (I_{\overline{C}} \cdot E)^*$. The base case, $B\ =\ \emptyset$, is easy. For the recursive case, we calculate:

$$\begin{array}{ll} & B \cdot (I_{\overline{C}} \cdot E)^* \\ = & \quad \{ \quad \text{breadth-first decomposition: lemma 16} \quad \} \\ & B \cup (B \cdot I_{\overline{C}} \cdot E) \cdot (I_{\overline{B}} \cdot I_{\overline{C}} \cdot E)^* \\ = & \quad \{ \quad I_{\overline{B}} \cdot I_{\overline{C}}\ =\ I_{\overline{B} \cap \overline{C}}\ =\ I_{\overline{B \cup C}} \quad \} \\ & B \cup (B \cdot I_{\overline{C}} \cdot E) \cdot (I_{\overline{B \cup C}} \cdot E)^* \\ = & \quad \{ \quad \bullet \quad R'(B\ ,\ C)\ =\ B \cdot (I_{\overline{C}} \cdot E)^* \quad \} \\ & B \cup R'(B \cdot I_{\overline{C}} \cdot E\ ,\ B \cup C) \end{array}$$

The termination argument is analogous to that for R. The set of visited nodes is non-decreasing: $B \cup C \supseteq C$, and if $B \cup C\ =\ C$ then $B \subseteq C$ and $B \cdot I_{\overline{C}}\ =\ B \cap \overline{C}\ =\ \emptyset$, so the terminating case is reached. Termination is guaranteed since the set of visited nodes is bounded by V , and $I_{\overline{V}}\ =\ 0$.

Notice that $B \cdot I_{\overline{C}} \cdot E\ =\ (B \cap \overline{C}) \cdot E\ =\ (B - C) \cdot E$. This final term is the one used in [Möl93].

4. Reachability: a relational approach

The definition of R' which we derived is essentially the same as the definition derived in [Möl93]. The derivation is different for three main reasons. First, Möller uses an algebra of relations based on sets of words rather than sets of pairs. This enables ordinary sets, i.e. sets of singleton words, and ordinary relations, i.e. sets of words of length 2, to be treated as the same sort, and handled uniformly under composition (but with qualified associativity). Second, Möller employs a path algebra which is closely related to his algebra of relations. Third, Möller proves lemma 16 (his corollary 5) by direct appeal to a closure induction principle. We now review Möller's approach and examine the significance of these differences.

Definition 17 (Words and languages) Given an alphabet, V , a *word* over V is a sequence of symbols drawn from V . The empty word is denoted ε. The set of all words over V is denoted V^*. A symbol $a \in V$ also denotes a singleton word $a \in V^*$. A language over V is a subset of V^*. A word $w \in V^*$ also denotes a singleton language $w \in \mathsf{P}V^*$. The set of non-empty words over V , $V^* - \varepsilon$, is denoted V^+. We note that Möller brackets * and + to make a distinction with the * and + as applied to relations. Here, we get away with overloading these symbols.

□

Definition 18 (Language, Relation, and Path algebras) The following algebras are exploited in [Möl93].

$$\mathit{LAN} \triangleq \langle \mathsf{P}V^*\ ,\ \cup\ ,\ \bullet\ ,\ \emptyset\ ,\ \varepsilon \rangle$$

$$\mathit{REL} \triangleq \langle \mathsf{P}(V \bullet V)\ ,\ \cup\ ,\ ;\ ,\ \emptyset\ ,\ I_V\ \rangle$$

$$PAT \triangleq \langle \mathsf{P}V^+ \ , \ \cup \ , \ \bowtie \ , \emptyset \ , \ V \ \rangle$$

Each of these algebras is obtained by lifting the following set-valued (remember, elements also stand for singleton sets) operations on words pointwise to operations on languages.

$$\begin{array}{lcl} u \bullet v & = & uv \\ ua \ ; \ bv & = & \begin{cases} uv & \text{if } a = b \\ \emptyset & \text{otherwise} \end{cases} \\ ua \bowtie bv & = & \begin{cases} uav & \text{if } a = b \\ \emptyset & \text{otherwise} \end{cases} \end{array}$$

It is easy to check that each of LAN , REL , and PAT is a Kleene algebra when * is added as the sum of all positive powers.

□

Möller points out that for $E \subseteq V \bullet V$, and $S \subseteq V$, $S \ ; E$ and $E \ ; S$ denote the image and inverse image, respectively, of S through E . So the reachability problem is captured by $S \ ; E^*$. But, of course, this set-by-relation composition, ;, is just our (Boolean) vector-by-matrix product, $\cdot$. More precisely, $\langle \mathsf{P}V \ , \ \mathsf{P}(V \bullet V) \ , \ ; \rangle$ is isomorphic to $\langle \mathbb{B}^V \ , \ \mathbb{B}^{V \times V} \ , \ \cdot \rangle$. Moreover, all of Möller's calculations use dynamic algebra specialised to relations, and they translate faithfully into our notation. His few uses of $\bowtie$ are easily eliminated by his own law (8), which states:

$$S \bowtie R \ = \ I_S \ ; R \ and \ R \bowtie S \ = \ R \ ; I_S$$

With this translation we can see that lemma 16 is Möller's corollary 5, and that the proof given here offers a shorter alternative.

5. Concluding remarks

In this paper we have highlighted the connection between two recent independent approaches to the formal calculation of graph algorithms, represented by [Möl93] and [BEG94]. That there is a connection is fairly obvious at an informal level, but since both approaches are concerned with formality, it is natural to ask for a formal explanation of the connection. We hope to have provided this explanation. In so doing we have appealed to the abstract setting of dynamic algebra. Dynamic algebra is more general than is needed to bring the two approaches together, but it encourages a convergence of terminology, and ties the work more deeply into established literature.

One might remark that path problems are privileged among the huge volume of graph-oriented problems since they enjoy good algebraic support. The algebraic calculation of algorithms for most other graph problems is still very much an open area of research. We propose, as might be expected, that this offers a fine test bed for further adventures with dynamic algebra. In particular, one wonders whether a useful body of re-usable lemmas and theorems might be established that progressively extends the applicability of dynamic algebra in graph algorithmics. Furthermore, the algebraic proofs are at a level of sufficient detail that they could be checked by machine, and this surely provides some motivation for using dynamic algebra as a test case for mechanised proof systems.

Acknowledgements I am grateful to Lambert Meertens for providing valuable feedback on an early version of this paper. Many thanks also to Andy Gordon and David King for their comments on a shorter version of this paper that appears as [Cle94].

This document was prepared using the MathSpad editing system developed at The University of Eindhoven, The Netherlands.

References

[BEG94] Roland C. Backhouse, J.P.H.W. van den Eijnde, and A.J.M. van Gasteren. Calculating path algorithms. *Science of Computer Programming*, 1994.

[Car79] Bernard Carré. *Graphs and Networks*. Clarendon Press, Oxford, 1979.

[Cle94] Kieran Clenaghan. Dynamic algebra for algorithm calculation. In Kevin Hammond and John T. O'Donnell, editors, *Functional Programming*, Workshops in Computing. Springer verlag, 1994.

[GM84] Michel Gondran and Michel Minoux. *Graphs and Algorithms*. Wiley-Interscience, New York, 1984.

[Gol92] Jonathon S. Golan. *The Theory of Semirings with Applications in Mathematics and Computer Science*. Longman Scientific and Technical, 1992.

[Heb90] Udo Hebisch. The Kleene theorem in countably complete semirings. *Bayreuther Mathematische Schriften*, 31:55–66, 1990.

[Kin90] Jeffrey H. Kingston. *Algorithms and Data Structures: Design, Correctness, and Analysis*. Addison-Wesley, 1990.

[Koz94] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2), 1994.

[KS86] Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1986.

[Leh77] Daniel J. Lehmann. Algebraic structures for transitive closure. *Theor. Comp. Sci.*, 4:59–76, 1977.

[Möl93] Bernhard Möller. Derivation of graph and pointer algorithms. Technical report, Institut fur Mathematik, Augsburg, 1993.

[MR93] Bernhard Möller and Martin Russling. Shorter paths to graph algorithms. In R.S.Bird, C.C.Morgan, and J.C.P.Woodcock, editors, *Mathematics of Program Construction (Oxford 1992)*, LNCS 669, 1993. Springer Verlag.

[Pra90] V.R. Pratt. Dynamic algebras as a well-behaved fragment of relation algebras. In D.L. Pigozzi C.H. Bergman, R.D. Maddux, editor, *Algebraic Logic and Universal Algebra in Computer Science (Iowa 1988)*, LNCS 425, 1990. Springer-Verlag.

[SS93] Gunther Schmidt and Thomas Strölein. *Relations and graphs*. Springer-Verlag, 1993.

A. A Shortest Paths algorithm and two specialisations

In this appendix we present a brief calculation of Dijkstra's single-source shortest paths algorithm and look at two specialisations of it, one to the reachability algorithm derived from scratch in this paper, and the other to Moore's shortest paths algorithm derived from scratch in [MR93]. This further exemplifies the algebraic calculational style. It also highlights the choice between deriving an algorithm directly from scratch, and deriving it as a specialisation of a more general algorithm that we already have.

Definition 19 (Shortest Paths)
Let $G = (V, E)$ be a graph, with $E : V \times V \to \mathbb{N}_\infty$ the adjacency matrix that assigns an edge length to each pair of nodes. The element $\infty \notin \mathbb{N}$ is assigned to pairs of nodes that are not adjacent, and $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$. The (single-source) shortest path problem is to find the lengths[1] of the shortest paths from a given start node $s \in V$ to each other node. We can equip $\mathbb{N} \cup \{\infty\}$ to make the so-called

[1] To find the shortest paths themselves is an easy extension of the algorithm for finding the shortest path lengths.

min-plus semiring, $\langle \mathbb{N}_\infty \, , \mathsf{min} \, , \, + \, , \infty \, , 0 \rangle$. Then, by the usual definition of matrix multiplication adapted to this semiring, E^n records the lengths of shortest n-paths. To see this, first note that the identity matrix, I, over $\mathbb{N}_\infty$ has a 0 diagonal, and ∞ elsewhere. Thus $E^0 \triangleq I$ does indeed record the shortest 0-path lengths. The shortest 1-path lengths are given by E itself. Now consider the matrix multiplication $E^p \cdot E \; = \; E^{p+1}$:

$$(E^p \cdot E)_{ij} \triangleq \mathsf{min}_{k \in V} \; ((E^p)_{ik} + E_{kj}) \tag{20}$$

If we assume that E^p records the shortest p-path lengths then we see (by induction) that E^{p+1} records the shortest $(p+1)$-path lengths.

The matrix of shortest path lengths for paths involving any number of edges is obtained by taking the minimum of the shortest p-path lengths for all $p \in \mathbb{N}$. We denote this by E^*:

$$E^* \triangleq \mathsf{min}_{p \in \mathbb{N}} \; E^p \tag{21}$$

The shortest paths problem is now succinctly captured as the computation of $s \cdot E^*$, where s is identified with the $1 \times V$ vector that has 0 in position s and ∞ elsewhere. The multiplication has the effect of selecting the s^{th} row of E^*. More generally, s can be an arbitrary initial assignment of distances to vertices that we might interpret as fixed distances from some imaginary starting point not in V. This general perspective will be assumed in calculations below where we make no assumptions about s.

□

With the min-plus semiring, $\langle \mathbb{N}_\infty \, , \mathsf{min} \, , \, + \, , \infty \, , 0 \rangle$, we can set up the dynamic algebra $\langle \{0 \, , 1\}^V \, , \mathbb{N}_\infty^{V \times V} \, , \cdot \rangle$. The min-plus semiring is idempotent and an idempotent semiring has a natural partial order given by:

$$a \leq b \; \triangleq \; a + b \; = \; b$$

(idempotency makes this coincide with the usual difference order definition: $a \leq b \; \triangleq \; \exists c \; . \; a + c \; = \; b$). This ordering extends pointwise to vectors and matrices in the obvious way. In the min-plus semiring, $a \leq b$ is actually the converse of the usual ordering on the naturals. With shortest paths in mind, we read $a \leq b$ as "a is longer than b", and we read $a \leq 1$ as "a is longer than 0".

In the following calculation we use singleton sets. To save braces, we write k instead of $\{k\}$. Moreover, k also stands for the $1 \times V$ vector that has a 1 in position k and 0 elsewhere, and $k^\top$ stands for its transpose. Note that $I_k \; = \; k^\top \cdot k$.

A.1 Dijkstra's shortest paths algorithm

Here we give a calculation of Dijkstra's shortest paths algorithm using the notation of an anonymous idempotent semiring. The particular semiring for shortest paths is the min-plus semiring described above. We shall interpret the anonymous notation in this context for motivational purposes, but we are keen to stick to the general notation to emphasise the independence of the calculations from the specific min-plus algebra. We discover that the calculations do not go through for any idempotent semiring, but we are able to characterise the extra properties abstractly, and with these we have a new semiring classification that Lehmann [Leh77] called a *Dijkstra* semiring.

To compute $s \cdot E^*$ we consider an incremental approach, in which at some arbitrary point we know $s \cdot E^* \cdot I_A$ – call it w_A – and we want to calculate $s \cdot E^* \cdot I_{A \cup k}$, that is, $w_{A \cup k}$. Clearly, $w_\emptyset \; = \; 0$ and $w_V \; = \; s \cdot E^*$. We read w_A as "the (lengths[2] of the) shortest paths to nodes in A".

[2] We shall hereafter use "shortest paths" to mean "lengths of shortest paths."

$$
\begin{array}{ll}
 & w_{A\cup k} \\
= & \quad \{ \quad \text{definition of } w \quad \} \\
 & s \cdot E^* \cdot I_{A\cup k} \\
= & \quad \{ \quad \text{disjoint distribution} \quad \} \\
 & s \cdot E^* \cdot I_A + s \cdot E^* \cdot I_k \\
= & \quad \{ \quad \text{definition of } w \quad \} \\
 & w_A + s \cdot E^* \cdot I_k \\
= & \quad \{ \quad \text{*-unfold: (13)} \quad \} \\
 & w_A + s \cdot (1 + E^* \cdot I_A \cdot E) \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k \\
= & \quad \{ \quad \text{algebra, definition of } w \quad \} \\
 & w_A + (s + w_A \cdot E) \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k
\end{array}
$$

So, we have derived:

$$w_{A\cup k} \;=\; w_A + (s + w_A \cdot E) \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k$$

This tells us that $w_{A\cup k}$ differs from w_A only in the kth position, if at all. Let $v_A \;=\; s + w_A \cdot E$, so that we get:

$$w_{A\cup k} \;=\; w_A + v_A \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k$$

We are interested in simplifying $v_A \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k$. In the shortest paths interpretation, this says we can get to k by extending some path (whose length is given by v_A) by a path (whose length is given by $(I_{\overline{A}} \cdot E)^*$) that goes exclusively through nodes outside A to k. The "key insight" is that if we choose k to be that (or any, if there is more than one) node outside A that is "nearest" then no extending path going through other nodes can possibly give a shorter path. This means that we can do away with the term $(I_{\overline{A}} \cdot E)^*$. By "nearest" we mean that the v_A distance is shortest. We can formalise this condition by (using subscript notation):

$$\forall(j :\; j{\in}\overline{A} :\; (v_A)_j \leq (v_A)_k)$$

or (without subscript notation):

$$\forall(j :\; j{\in}\overline{A} :\; v_A \cdot j^\top \leq v_A \cdot k^\top)$$

For brevity, let $P \triangleq \forall(j :\; j{\in}\overline{A} :\; (v_A)_j \leq (v_A)_k)$. In subsections A.2 and A.3 we establish the precise conditions that enable us to prove:

$$v_A \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k \;=\; v_A \cdot I_k \;\Leftarrow\; P$$

Using this fact we get:

$$w_\emptyset \;=\; 0$$

$$w_{A\cup k} \;=\; w_A + v_A \cdot I_k \;\Leftarrow\; P$$

We are not done, the complexity is O($\mid V \mid^3$) since the calculation of v_A on each iteration is O($\mid V \mid^2$). This motivates us to consider calculating v_A incrementally. We note that $v_\emptyset \;=\; s + w_\emptyset \cdot E \;=\; s$, and we look to define $v_{A\cup k}$ in terms of v_A:

$$\begin{array}{ll} & v_{A\cup k} \\ = & \quad \{ \quad \text{definition of } v \quad \} \\ & s + w_{A\cup k} \cdot E \\ = & \quad \{ \quad \text{definition of } w \quad \} \\ & s + (w_A + v_A \cdot I_k) \cdot E \\ = & \quad \{ \quad \text{algebra} \quad \} \\ & s + w_A \cdot E + (v_A \cdot I_k \cdot E) \\ = & \quad \{ \quad \text{definition of } v \quad \} \\ & v_A + (v_A \cdot I_k \cdot E) \end{array}$$

This is much better because $v_A \cdot I_k \cdot E$ (and the choice of k) can be computed in O($\mid V \mid$). Moreover, we have:

$$v_V \;=\; s + w_V \cdot E \;=\; s + s \cdot E^* \cdot E \;=\; s \cdot E^*$$

so we can do without w altogether.

$$v_\emptyset \;=\; s$$

$$v_{A\cup k} \;=\; v_A + v_A \cdot I_k \cdot E \;\Leftarrow\; P$$

We should point out that [BEG94] refine this algorithm further. In particular, it is proved that the second equation can be separated into the following two equations (a fact which is exploited later in the specialisation to shortest edge-counts, section A.5).

$$v_{A\cup k} \cdot I_{A\cup k} \;=\; v_A \cdot I_{A\cup k}$$

$$v_{A\cup k} \cdot I_{\overline{A\cup k}} \;=\; (v_A + v_A \cdot I_k \cdot E) \cdot I_{\overline{A\cup k}} \Leftarrow P$$

A.2 The key insight of Dijkstra's algorithm
Here we consider the precise conditions under which the "key insight",

$$v_A \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k \;=\; v_A \cdot I_k \;\Leftarrow\; P$$

holds, where $P \triangleq \forall(j :\; j \in \overline{A} :\; v_A \cdot j^\top \leq v_A \cdot k^\top)$.

We start by quickly reducing $v_A \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k \;=\; v_A \cdot I_k$ to an inequality, because:

$$
\begin{array}{ll}
 & v_A \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k \\
\geq & \quad \{ \quad a^* \geq 1 \quad \} \\
 & v_A \cdot I_k
\end{array}
$$

It remains to show the converse:

$$v_A \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k \leq v_A \cdot I_k$$

In order to apply P we wish to expose a $v_A \cdot j^\top$ term. Observe:

$$
\begin{array}{ll}
 & v_A \cdot I_{\overline{A}} \\
= & \quad \{ \quad \text{definition of } I_{\overline{A}} \text{ and } \cdot \quad \} \\
 & \sum_{j \in \overline{A}} v_A \cdot j^\top \cdot j
\end{array}
$$

Therefore, if we expose a $v_A \cdot I_{\overline{A}}$ term then we can make use of P:

$$
\begin{array}{ll}
 & v_A \cdot (I_{\overline{A}} \cdot E)^* \cdot I_k \\
= & \quad \{ \quad I_k \ = \ I_{\overline{A}} \cdot I_k \quad \} \\
 & v_A \cdot (I_{\overline{A}} \cdot E)^* \cdot I_{\overline{A}} \cdot I_k \\
= & \quad \{ \quad \text{leap-frog} \quad \} \\
 & v_A \cdot I_{\overline{A}} \cdot (E \cdot I_{\overline{A}})^* \cdot I_k \\
= & \quad \{ \quad \text{definitions of } I_{\overline{A}} \text{ and } I_k \quad \} \\
 & (\sum_{j \in \overline{A}} v_A \cdot j^\top \cdot j) \cdot (E \cdot I_{\overline{A}})^* \cdot k^\top \cdot k \\
= & \quad \{ \quad \text{distributivity} \quad \} \\
 & \sum_{j \in \overline{A}} v_A \cdot j^\top \cdot j \cdot (E \cdot I_{\overline{A}})^* \cdot k^\top \cdot k \\
\leq & \quad \{ \quad P \quad \} \\
 & \sum_{j \in \overline{A}} v_A \cdot k^\top \cdot j \cdot (E \cdot I_{\overline{A}})^* \cdot k^\top \cdot k
\end{array}
$$

The term $j \cdot (E \cdot I_{\overline{A}})^* \cdot k^\top$ denotes an element. In the shortest paths interpretation it is the shortest distance from j via an $\overline{A}$-path to k. Any distance is at least 0, and by the min-plus interpretation this is expressed as $\forall a.a \leq 1$ in the general semiring notation. Using this we complete our proof:

$$
\begin{array}{ll}
 & \sum_{j \in \overline{A}} v_A \cdot k^\top \cdot j \cdot (E \cdot I_{\overline{A}})^* \cdot k^\top \cdot k \\
\leq & \quad \{ \quad j \cdot (E \cdot I_{\overline{A}})^* \cdot k^\top \leq 1 \text{ and monotonicity} \quad \} \\
 & \sum_{j \in \overline{A}} v_A \cdot k^\top \cdot k \\
= & \quad \{ \quad \text{definition of } I_k \quad \} \\
 & v_A \cdot I_k
\end{array}
$$

Notice that proposition 14 is a special case of the "key insight".

A.3 The Dijkstra semiring
The "key insight" of Dijkstra's algorithm cannot be applied in any idempotent semiring. First, we note that it must be totally ordered (since arbitrary elements are compared), and second, we need to have 1 as the largest element as indicated in the calculations above. [Leh77] calls semirings satisfying $\forall a.a \leq 1$ *simple*, because * becomes simple: $a^* = 1$. A simple semiring is necessarily idempotent (easy exercise). [Leh77] calls a totally ordered simple semiring a *Dijkstra* semiring.

If we name Dijkstra's algorithm SP, then we can express its parameterisation with respect to a *Dijkstra* semiring symbolically by $SP(S{:}\mathit{DijkstraSemiring})$. In the next two sections we consider special instances for S.

A.4 Reachability as a special case of shortest paths
Recall Dijkstra's algorithm which we named SP in the previous section:

$$v_\emptyset \;=\; s$$

$$v_{A\cup k} \;=\; v_A + v_A \cdot I_k \cdot E \;\Leftarrow\; \forall(j:\; j{\in}\overline{A}:\; (v_A)_j \leq (v_A)_k)$$

The instantiation $SP(\langle \mathbb{B}\ ,\ \vee\ ,\ \wedge\ ,0\ ,1\rangle)$ yields an algorithm that solves the reachability problem. Here is its expansion, using C instead of A (C is used in the reachability algorithm of section 3).

$$v_\emptyset \;=\; S$$

$$v_{C\cup k} \;=\; v_C \cup v_C \cdot I_k \cdot E \;\Leftarrow\; \forall(j:\; j{\in}\overline{C}:\; (v_C)_j \leq (v_C)_k)$$

With this instantiation, no advantage is taken of the special properties of the Boolean semiring. Here we show how consideration of these special properties can lead to (an iterative version of) the reachability algorithm of section 3 and [Möl93].

In the reachability interpretation, v_C is the set of C-reachable nodes. The choice of k such that $\forall(j:\; j{\in}\overline{C}:\; (v_C)_j \leq (v_C)_k)$ is just the preference of a C-reachable $k{\in}\overline{C}$ over a non-C-reachable one. The set $v_C{\cap}\overline{C} \;=\; v_C \cdot I_{\overline{C}}$ is precisely the set of all C-reachable nodes in $\overline{C}$. We can express the condition that k is in this set by:

$$k \;=\; v_C \cdot I_{\overline{C}} \cdot I_k$$

Asking for this to hold is stronger than the preference $\forall(j:\; j{\in}\overline{C}:\; (v_C)_j \leq (v_C)_k)$. But, if $v_C \cdot I_{\overline{C}}$ is empty and we choose $k{\in}\overline{C}$, then $v_{C\cup k} \;=\; v_C$. With this in mind we concentrate on the non-empty case:

$$v_\emptyset \;=\; S$$

$$v_{C\cup k} \;=\; v_C \cup v_C \cdot I_k \cdot E \;\Leftarrow\; k \;=\; v_C \cdot I_{\overline{C}} \cdot I_k$$

We also have that $k \;=\; v_C \cdot I_{\overline{C}} \cdot I_k \;\Rightarrow\; k \;=\; v_C \cdot I_k$, hence:

$$v_\emptyset \;=\; S$$

$$v_{C \cup k} \;=\; v_C \cup k \cdot E \;\Leftarrow\; k \;=\; v_C \cdot I_{\overline{C}} \cdot I_k$$

By repeated application of the second equation for all possible choices of k we get:

$$v_\emptyset \;=\; S$$

$$k_1 \cup ... \cup k_n \;=\; v_C \cdot I_{\overline{C}} \;\Rightarrow$$

$$v_{C \cup k_1 \cup ... \cup k_n} \;=\; v_C \cup k_1 \cdot E \cup ... \cup k_n \cdot E$$

And, by distributivity:

$$v_\emptyset \;=\; S$$

$$v_{C \,\cup\, v_C \,\cdot\, I_{\overline{C}}} \;=\; v_C \cup v_C \cdot I_{\overline{C}} \cdot E$$

To proceed to relate this definition to the definition of R' in section 3 we use ideas from transformational programming. Our approach is to show that both can be converted into the same "iterative" form. We start by converting the definition above to:

$$T(v\ ,\ C) \;=\; T(v \cup v \cdot I_{\overline{C}} \cdot E \;\;,\;\; C \cup v \cdot I_{\overline{C}}) \;\Leftarrow\; v \cdot I_{\overline{C}} \;\neq\; \emptyset$$

$$T(v\ ,\ C) \;=\; v \;\Leftarrow\; v \cdot I_{\overline{C}} \;=\; \emptyset$$

We think of T as a loop with variables v and C. The "initialisation" is $T(S\ ,\ \emptyset)$. Now, recall R' from section 3:

$$R'(\emptyset\ ,\ C) \;=\; \emptyset$$

$$R'(B\ ,\ C) \;=\; B \cup R'(B \cdot I_{\overline{C}} \cdot E\ ,\ B \cup C)$$

We'll turn this into an iterative form by introducing an accumulating variable, v:

$$R''(v\ ,\ B\ ,\ C) \;=\; v \cup R'(B \cdot I_{\overline{C}} \cdot E\ ,\ B \cup C)$$

We can eliminate R' (we use the second equation of R' – it holds even when $B \;=\; \emptyset$):

$$\begin{array}{ll} & R''(v\ ,\ B\ ,\ C) \\ = & \qquad \{ \qquad \text{definition of } R'' \quad \} \\ & v \cup R'(B \cdot I_{\overline{C}} \cdot E\ ,\ B \cup C) \end{array}$$

$$
\begin{array}{ll}
= & \{ \quad \text{definition of } R' \quad \} \\
& v \cup B \cdot I_{\overline{C}} \cdot E \cup R'(B \cdot I_{\overline{C}} \cdot E \cdot I_{\overline{B \cup C}} \cdot E \ , \ B \cdot I_{\overline{C}} \cdot E \cup B \cup C) \\
= & \{ \quad \text{definition of } R'' \quad \} \\
& R''(v \cup B \cdot I_{\overline{C}} \cdot E \ , \ B \cdot I_{\overline{C}} \cdot E \ , \ B \cup C)
\end{array}
$$

The terminating condition is $B = \emptyset$, and $R''(S\ ,\ S\ ,\ \emptyset)$ computes $S \cdot E^*$. We wish to show that $R''(S\ ,\ S\ ,\ \emptyset)$ can be transformed into $R'''(S\ ,\ \emptyset)$ such that $R''' = T$. This is straightforward if we can replace $B \cdot I_{\overline{C}}$ by $v \cdot I_{\overline{C}}$ and $B \cup C$ by $v \cdot I_{\overline{C}} \cup C$. The latter replacement follows from the former since (by set theory):

$$B \cup C \ = \ B \cdot I_{\overline{C}} \cup C$$

So, we need only show $B \cdot I_{\overline{C}} = v \cdot I_{\overline{C}}$ as an invariant of R''. It obviously holds for $R''(S\ ,\ S\ ,\ \emptyset)$ since $B = v$. The inductive case is calculated as follows:

$$
\begin{array}{ll}
& (v \cup B \cdot I_{\overline{C}} \cdot E) \cdot I_{\overline{B \cup C}} \\
= & \{ \quad \text{distributivity} \quad \} \\
& v \cdot I_{\overline{B \cup C}} \cup B \cdot I_{\overline{C}} \cdot E \cdot I_{\overline{B \cup C}} \\
= & \{ \quad \text{De Morgan, commutativity.} \quad \} \\
& v \cdot I_{\overline{C}} \cdot I_{\overline{B}} \cup B \cdot I_{\overline{C}} \cdot E \cdot I_{\overline{B \cup C}} \\
= & \{ \quad \text{ind. hyp.} \quad \} \\
& B \cdot I_{\overline{C}} \cdot I_{\overline{B}} \cup B \cdot I_{\overline{C}} \cdot E \cdot I_{\overline{B \cup C}} \\
= & \{ \quad \text{commutativity} \quad \} \\
& B \cdot I_{\overline{B}} \cdot I_{\overline{C}} \cup B \cdot I_{\overline{C}} \cdot E \cdot I_{\overline{B \cup C}} \\
= & \{ \quad B \cdot I_{\overline{B}} = \emptyset \quad \} \\
& B \cdot I_{\overline{C}} \cdot E \cdot I_{\overline{B \cup C}}
\end{array}
$$

This means we can eliminate B from the definition of R'' to obtain R''':

$$R'''(v\ ,\ C) \ = \ R'''(v \cup v \cdot I_{\overline{C}} \cdot E \ , \ v \cdot I_{\overline{C}} \cup C)$$

But, with the addition of the terminating condition, this is just T, and we are done.

A.5 Specialisation to shortest edge counts

Möller's calculational style is further illustrated in [MR93] where an algorithm is calculated for computing the smallest number of edges between two vertices. The same problem is solved by $SP(\langle \mathbb{N} \cup \{\infty\}\ ,\ \mathsf{min}\ ,\ +\ ,\ \infty\ ,\ 0\rangle)$ applied to a matrix whose elements are 0 (reflexive edge), 1 (direct edge), or ∞ (no edge). Therefore it is natural to consider specialising SP as opposed to calculating an algorithm from scratch for the specialised problem. We shall do this here to a limited extent. We start with a brief overview of the approach in [MR93] and then remark on specialising SP.

The calculation in [MR93] starts from scratch with the following specification:

$$sp0(x\ ,\ y) \triangleq min(edgecount(x \ \bowtie \ E^* \ \bowtie \ y))$$

where E^* is the path-closure of E, *edgecount* gives the number of edges in a path, and *min* selects the minimum value in a set. Some economy is achieved by defining functions (including constants) to be set-valued, and silently lifting them (à la Kleisli) to take set-valued arguments. In this way *edgecount* applies to a set of paths. No unit (e.g. ∞) is stipulated for *min*, and instead, *min* is defined to map the empty set to the empty set. Hence *sp0* is set-valued, returning the empty set if there is no path from x to y, otherwise the singleton set containing the length of the shortest path.

The specification *sp0* is quickly generalised to permit a set, S, of starting nodes:

$$sp(S\ ,\ y) \triangleq min(edgecount(S\ \bowtie\ E^*\ \bowtie\ y))$$

The instance $S\ =\ x$ recovers the original specification (elements are overloaded as singleton sets).

The derivation of an algorithm for *sp* proceeds by calculation, appealing to laws from relational algebra and properties of *min* and *edgecount*. As is the derivation of the reachability algorithm in [Möl93] a special induction principle is unnecessarily introduced and applied. The algorithm finally obtained is the following[3], in which an extra argument, T, keeps track of vertices already visited.

$$sp3(S\ ,T\ ,\ y)\ =\ 0 \Leftarrow y{\in}S$$

$$sp3(S\ ,T\ ,\ y)\ =\ \emptyset\ \ \Leftarrow\ \ y{\notin}S\ \wedge\ S\ ;R \subseteq S{\cup}T$$

$$y{\notin}S\ \wedge\ S\ ;R\ \supset\ S{\cup}T\ \Rightarrow$$

$$sp3(S\ ,T\ ,\ y)\ =\ 1{+}sp3((S\ ;R){-}(S{\cup}T)\ ,\ (S{\cup}T)\ \ ,\ \ y)$$

The main result of the derivation is:

$$sp0(x\ ,\ y)\ =\ sp3(x\ ,\ \emptyset\ ,\ y)$$

Now we have a look at specialising Dijkstra's algorithm SP (with a view to getting close to *sp3*). Recall SP:

$$v_\emptyset\ =\ s$$

$$v_{C\cup k}\ \ =\ \ v_C\ +\ v_C\cdot I_k\cdot E\ \Leftarrow\ \forall(j:\ j{\in}\overline{C}:\ (v_C)_j \leq (v_C)_k)$$

In the shortest edge-count interpretation, v_C records the shortest C-paths to all nodes, where a path length is the number of edges in the path. As in the reachability interpretation, we can increase C by choosing any, or all, $\overline{C}$ nodes that are C-reachable. The set of C-reachable nodes are precisely those with non-∞ paths in v_C. We denote this set by $\overrightarrow{v_C}$, which we obtain from v_C by replacing each 1 by 0. With this, and taking the start vertex to be x, our first specialisation of SP is:.

$$v_\emptyset\ =\ x$$

[3]Some invariants crucial to the correctness of this algorithm are omitted.

$$v_{C\cup B} \quad = \quad (v_C \ \mathtt{min}\ v_C \cdot I_B \cdot E) \Leftarrow B = \overrightarrow{v_C} \cdot I_{\overline{C}}$$

For any $j \in C\cup B$, v_C already records the shortest path length to j, so we can restrict attention to $j \notin C\cup B$ (see the remark at the end of section A.1): For $j \notin C\cup B$, $v_C \cdot j^\top = \infty$, so we have:

$$B = \overrightarrow{v_C} \cdot I_{\overline{C}} \Rightarrow$$

$$v_{C\cup B} \cdot I_{C\cup B} = v_C \cdot I_{C\cup B} \textit{ and}$$

$$v_{C\cup B} \cdot I_{\overline{C\cup B}} = v_C \cdot I_B \cdot E \cdot I_{\overline{C\cup B}}$$

Expanding the right-hand-side of the second equation for $j \notin C\cup B$ we get:

$$v_C \cdot I_B \cdot E \cdot j^\top = \mathsf{min}_{k\in B}\ (v_C \cdot I_B \cdot k^\top)+(k \cdot E \cdot j^\top)$$

The element $k \cdot E \cdot j^\top$ is either 1 or ∞, since $k \neq j$. By the equations:

$$(a+1)\ \mathsf{min}\ (b+1) = 1+((a+0)\ \mathsf{min}\ (b+0))$$

$$\infty = \infty+1$$

we can transform E into a connectivity relation, $\overrightarrow{E}$, by replacing each 1 by 0, and get:

$$B = \overrightarrow{v_C} \cdot I_{\overline{C}} \Rightarrow$$

$$v_{C\cup B} \cdot I_{\overline{C\cup B}} = (1 + v_C \cdot I_B \cdot \overrightarrow{E}) \cdot I_{\overline{C\cup B}}$$

This takes us far enough to see some of the formal issues in refining *SP* to take advantage of the special properties of the edge-count problem. It has not reduced *SP* to *sp3*. To do that requires more transformations which we omit in favour of some informal remarks (the reader may like to fill in the formal details as an exercise).

In proceeding to reconcile the definitions of v_C and *sp3*, we face the problem that v_C defines a vector, whereas *sp3* defines a scalar. This is because v_C gives the shortest C-paths to all vertices, not just to a particular vertex, y, as in *sp3*. By restricting to a target vertex, y, we can reduce v_C to a scalar since, either $y \in C\cup B$ and we are done, or $y \notin C\cup B$ and we have a uniform increment of 1 on the lengths of paths to B-nodes (i.e. C-reachable $\overline{C}$-nodes) which extend by an edge to $\overline{C\cup B}$-nodes. Initially, the B-nodes are the $\emptyset$-reachable nodes, of which there is just one, x, with path-length 0,. Therefore, by induction, the uniformity of the increment maintains the equality of the path lengths to the reachable $\overline{C\cup B}$-nodes, and so we need record only one length. This means that we can interpret v_C as a scalar, being the shortest C-path length to any node in $C\cup B$. But, we can no longer deduce B from v_C, so we carry it explicitly by writing $v_{B\ ,\ C}$.

$$v_{x\ ,\ \emptyset} = 0$$

$$B' \;=\; I_B \cdot \overrightarrow{E} \cdot I_{\overline{C \cup B}} \;\neq\; \emptyset \;\Rightarrow$$

$$v_{B'\ ,\ C \cup B} \quad = \quad 1 + v_{B\ ,\ C}$$

It is easy now to match this up with the definition of *sp3*. B is S, C is T, $\overrightarrow{E}$ is R, hence,

$$I_B \cdot \overrightarrow{E} \cdot I_{\overline{C \cup B}} \;=\; S\ ;\ R - (T \cup S)$$

$$I_B \cdot \overrightarrow{E} \cdot I_{\overline{C \cup B}} \;\neq\; \emptyset \;\equiv\; S\ ;\ R \;\supset\; T \cup S$$

With these identities we convert the definition of $v_{B\ ,\ C}$ to iterative form, sp':

$$S' \;=\; S\ ;\ R - (T \cup S) \quad \neq \quad \emptyset \;\Rightarrow$$

$$sp'(v\ ,\ S\ ,\ T) \;=\; sp'(1{+}v\ ,\ S'\ ,\ (T \cup S))$$

The initialisation is $sp'(0\ ,\ x\ ,\ \emptyset)$. Similarly, *sp3* can be converted to iterative form by introducing an accumulating variable: $sp3(v\ ,\ x\ ,\ \emptyset\ ,\ y)$. By introducing y into sp' and adding the terminating cases we can complete the tie-up.