Layered Graph Traversals and Hamiltonian Path Problems – An Algebraic Approach

Thomas Brunn[1], Bernhard Möller[2], and Martin Russling[3]

[1] Feilmeier, Junker & Co. (Institut für Wirtschafts- und Versicherungsmathematik GmbH), München
[2] Institut für Informatik, Universität Augsburg
[3] R. Böker Unternehmensgruppe AG, München

Abstract. Using an algebra of paths we present abstract algebraic derivations for two problem classes concerning graphs, viz. layer oriented traversal and computing sets of Hamiltonian paths. In the first case, we are even able to abstract to the very general setting of Kleene algebras. Applications include reachability and a shortest path problem as well as topological sorting and finding maximum cardinality matchings.

1 Introduction

Starting out from ideas in [Möller 91] the paper [Möller, Russling 93] presented algebraic derivations of a few graph algorithms. While there the algorithms were treated separate from each other, in [Russling 96a,Russling 96b] a systematization into classes of graph algorithm was achieved. For each class a schematic algorithm was derived; the problems in the class are then solvable by instantiations of that schematic algorithm.

This topic has been further pursued in [Brunn 97], which contains two innovations. In the case of layer-oriented graph traversal the case of a set of starting *nodes* was generalized to that of a set of starting *paths*. This allowed dropping one essential assumption in the earlier developments, made the overall derivation smoother and led to a simplified algorithm. Second, in the case of Hamiltonian path problems, a significant new application of the general scheme was found, viz. that of computing maximum cardinality matchings.

Next to presenting the above-mentioned two classes and some of their instances, in the present paper we are able to abstract the derivation of the layer-oriented traversal algorithm to typed Kleene algebras; even the efficiency improvement can be performed at this very abstract level.

2 Graphs and Path Languages

2.1 Formal Languages and Relations

Consider a finite alphabet A. The set of all words over A is denoted by A^* and the empty word by ε. A *(formal) language* V is a subset of A^*. For simplicity

This research was partially funded by Esprit Working Group 8533 — NADA: New Hardware Design Methods

we identify a singleton set with its only element and a word of length 1 with its only letter. The set of non-empty words is $A^+ \stackrel{\text{def}}{=} A^* \backslash \varepsilon$. Concatenation is denoted by $\bullet$; it is associative and has ε as its neutral element.

In connection with graph algorithms the letters of A are interpreted as nodes. The words of a language are used to represent paths in that graph: the nodes are listed in the order of traversal. The *length* of a word w is the number of letters in w and is denoted by $\|w\|$.

A *relation* is a language R in which all words have equal length. This length is called the *arity* of the relation, in symbols $\mathsf{ar}\, R$. The empty relation $\emptyset$ has all arities. Unary relations can be interpreted as sets of nodes, whereas binary relations represent sets of edges. The binary *identity relation* is

$$I \stackrel{\text{def}}{=} \{a \bullet a : a \in A\} \ ,$$

whereas $A \bullet A$ is the *universal relation* on A.

2.2 Pointwise Extension

The operations on languages we are going to define in the next section will first be explained for single words and then lifted pointwise to languages.

The *pointwise extension* of an operation $f : A^* \to \mathcal{P}(A^*)$ is denoted by the same symbol f and has the functionality $f : \mathcal{P}(A^*) \to \mathcal{P}(A^*)$. It is defined by

$$f(U) \stackrel{\text{def}}{=} \bigcup_{x \in U} f(x)$$

for $U \subseteq A^*$. This definition implies that the extension is *universally disjunctive*, i.e., distributes over arbitrary unions, is *strict* w.r.t. $\emptyset$, i.e., satisfies $f(\emptyset) = \emptyset$, and is monotonic w.r.t. $\subseteq$.

Moreover, linear laws for f, i.e., equational laws in which all variables occur exactly once on both sides of the equality sign, are inherited by the pointwise extension.

The pointwise extension generalizes in a straightforward way to operations with more than one argument.

2.3 Join and Composition

For words s and t over alphabet A we define their **join** $s \bowtie t$ and their **composition** $s \,;\, t$ by

$$\varepsilon \bowtie s \;=\; \emptyset \;=\; s \bowtie \varepsilon \ , \qquad \varepsilon \,;\, s \;=\; \emptyset \;=\; s \,;\, \varepsilon \ ,$$

and, for $s, t \in A^*$ and $x, y \in A$, by

$$(s \bullet x) \bowtie (y \bullet t) \stackrel{\text{def}}{=} \begin{cases} s \bullet x \bullet t \text{ if } x = y \ , \\ \emptyset \qquad \text{otherwise} \ , \end{cases}$$

$$(s \bullet x) \ ; \ (y \bullet t) \stackrel{\text{def}}{=} \begin{cases} s \bullet t \text{ if } x = y \ , \\ \emptyset \quad \text{otherwise} \ . \end{cases}$$

These operations provide two different ways of "gluing" two words together upon a one-letter overlap: join preserves one copy of the overlap, whereas composition erases it. Again, they are extended pointwise to languages. On binary relations, composition coincides with usual relational composition (see e.g. [Schmidt, Ströhlein 93]). To save parentheses we use the convention that $\bullet$, $\bowtie$ and ; bind stronger than all set-theoretic operations.

To exemplify the close connection between join and composition further, we consider a binary relation $R \subseteq A \bullet A$ modeling the edges of a directed graph with node set A. Then

$$\begin{aligned} R \bowtie R &= \{x \bullet z \bullet y \ : \ x \bullet z \in R \wedge z \bullet y \in R\} \ , \\ R \,;\, R &= \{x \bullet y \ : \ x \bullet z \in R \wedge z \bullet y \in R\} \ . \end{aligned}$$

Thus, the relation $R \bowtie S$ consists of exactly those paths $x \bullet z \bullet y$ which result from gluing two edges together at a common intermediate node. The composition $R \,;\, R$ is an abstraction of this; it just states whether there is a path from x to y via some intermediate point without making that point explicit. Iterating this observation shows that the relations

$$R, \ R \bowtie R, \ R \bowtie (R \bowtie R), \ \ldots$$

consist of the paths with exactly $1, 2, 3, \ldots$ edges in the directed graph associated with R, whereas the relations

$$R, \ R \,;\, R, \ R \,;\, (R \,;\, R), \ \ldots$$

just state existence of these paths between pairs of vertices.

The pointwise extension of join yields the following result for unary relations S and T:

$$S \bowtie T = S \cap T \ . \tag{1}$$

Finally, we have the associativities (see [Möller 93] for further ones)

$$\left.\begin{aligned} U \bowtie (V \bowtie W) &= (U \bowtie V) \bowtie W \ , \\ U \bullet (V \bowtie W) &= (U \bullet V) \bowtie W \quad \Leftarrow V \cap \varepsilon = \emptyset \ , \\ U \bowtie (V \bullet W) &= (U \bowtie V) \bullet W \quad \Leftarrow V \cap \varepsilon = \emptyset \ , \end{aligned}\right\} \tag{2}$$

3 Kleene Algebras and Closures

3.1 Kleene Algebras

A *Kleene algebra* (cf. [Conway 71]) is a quintuple $(S, \Sigma, \cdot, 0, 1)$ consisting of a set S, operations $\Sigma : \mathcal{P}(S) \to S$ and $\cdot : S \bullet S \to S$ as well as elements $0, 1 \in S$ such that $(S, \cdot, 1)$ is a monoid and

$$\begin{aligned} \Sigma\, \emptyset &= 0 \ , \\ \Sigma\, \{x\} &= x && (x \in S) \ , \\ \Sigma(\cup \mathcal{K}) &= \Sigma\, \{\Sigma\, K : K \in \mathcal{K}\} && (\mathcal{K} \subseteq \mathcal{P}(S)) \ , \\ \Sigma\, (K \cdot L) &= (\Sigma\, K) \cdot (\Sigma\, L) && (K, L \in \mathcal{P}(S)) \ , \end{aligned}$$

where $\cdot$ in the latter equation is the pointwise extension of the original $\cdot$ operation. The definition implies that $\cdot$ is strict w.r.t. 0:

$$0 \cdot x \;=\; 0 \;=\; x \cdot 0 \;\;.$$

In connection with graph algorithms one often considers the related structure of a *closed semiring* (see e.g. [Aho et al. 74]). It differs from a Kleene algebra in that ΣK is only required to exist for *countable* K; moreover, idempotence of $+$ is not postulated. So every Kleene algebra is a closed semiring, but not vice versa.

Examples of Kleene algebras are given by

$$\begin{aligned} LAN &\stackrel{\text{def}}{=} (\mathcal{P}(A^*),\ \bigcup,\ \bullet,\ \emptyset,\ \varepsilon) \;\;, \\ REL &\stackrel{\text{def}}{=} (\mathcal{P}(A \bullet A),\ \bigcup,\ ;,\ \emptyset,\ I) \;\;, \\ PAT &\stackrel{\text{def}}{=} (\mathcal{P}(A^*),\ \bigcup,\ \bowtie,\ \emptyset,\ A \cup \varepsilon) \;\;. \end{aligned}$$

For a Kleene algebra one can define a partial order as follows:

$$x \leq y \;\stackrel{\text{def}}{\Leftrightarrow}\; x + y \;=\; y \;\;, \tag{3}$$

where

$$x + y \;\stackrel{\text{def}}{=}\; \Sigma\{x, y\} \;\;. \tag{4}$$

Then $(S, \leq)$ forms a complete lattice. We denote the greatest element of that lattice by $\top$. In the above examples $\leq$ coincides with $\subseteq$.

Let μ denote the least fixpoint operator in a complete lattice. With its help we can define an improper closure operator $.^*$ by

$$x^* \;\stackrel{\text{def}}{=}\; \mu\, y \,.\, 1 + x \cdot y \;\;, \tag{5}$$

and a proper closure operator $.^+$ by

$$x^+ \;\stackrel{\text{def}}{=}\; \mu\, y \,.\, x + x \cdot y \;\;.$$

3.2 Path Closure

For a directed graph with node set A and edge set $R \subseteq A \bullet A$ we define the *path closure* $R^{\rightsquigarrow}$ to be the improper closure R^* in the Kleene algebra PAT. Since $\bowtie$ is universally disjunctive and hence chain-continuous, we know from [Kleene 52] that

$$R^{\rightsquigarrow} = \bigcup_{i \in \mathbb{N}} {}^iR \;\;,$$

where

$${}^0R \;\stackrel{\text{def}}{=}\; A \cup \varepsilon \;\;, \tag{6}$$

$${}^{i+1}R \;\stackrel{\text{def}}{=}\; R \bowtie {}^iR \;\;. \tag{7}$$

Hence ${}^iR$ is the relation consisting of all paths of length i.

It should be mentioned that the closures R^* and R^+ of a binary relation in REL are the reflexive-transitive closure and transitive closure, resp.

3.3 Typed Kleene Algebras

A *subidentity* of a Kleene algebra is an element x with $x \leq 1$. We call a Kleene algebra *pre-typed* if all its subidentities are idempotent, i.e., if $x \leq 1 \Rightarrow x \cdot x = x$. The subidentities then play the role of types. Moreover, restriction and co-restriction of an element a to a subtype x are given by $x \cdot a$ and $a \cdot x$, resp. We have

$$x, y \leq 1 \Rightarrow x \cdot y = x \sqcap y \ ,$$

i.e., the infimum of two types is their product.

We call a pre-typed Kleene algebra *typed* if it is a boolean algebra and the restriction operations distribute through arbitrary meets of subtypes, i.e., if we have for all sets K of subidentities and all $a \in S$ that

$$(\sqcap K) \cdot a = \sqcap(K \cdot a) \ \wedge \ a \cdot (\sqcap K) = \sqcap(a \cdot K) \ .$$

Then the subidentities are called *types*.

In a typed Kleene algebra we can define, for $a \in S$, the *domain* $\ulcorner a$ and *co-domain* $a\urcorner$ via the Galois connections (y ranges over subidentities only!)

$$\begin{aligned} \ulcorner a \leq y &\stackrel{\text{def}}{\Leftrightarrow} a \leq y \cdot \top \ , \\ a\urcorner \leq y &\stackrel{\text{def}}{\Leftrightarrow} a \leq \top \cdot y \ . \end{aligned}$$

By this, the operations $\ulcorner\_$ and $\_\urcorner$ are universally disjunctive and hence monotonic and strict. Moreover, we can show the usual properties of domain and co-domain (see [Möller 98]):

$$\left.\begin{aligned} \ulcorner a &= \sqcap\{x : x \leq 1 \wedge x \cdot a = a\} \ , \\ a\urcorner &= \sqcap\{y : y \leq 1 \wedge a \cdot y = a\} \ , \\ x \leq 1 &\Rightarrow \ulcorner x = x = x\urcorner \ , \\ \ulcorner(\ulcorner a) &= \ulcorner a \ , & (a\urcorner)\urcorner &= a\urcorner \ , \\ \ulcorner(a \cdot b) &= \ulcorner(a \cdot \ulcorner b) \ , & (a \cdot b)\urcorner &= (a\urcorner \cdot b)\urcorner \ , \\ \ulcorner(\ulcorner a \cdot b) &= \ulcorner a \sqcap \ulcorner b \ , & (a \cdot b\urcorner)\urcorner &= a\urcorner \sqcap b\urcorner \ , \\ \ulcorner a \leq \ulcorner b \Rightarrow \ulcorner(a \cdot c) &\leq \ulcorner(b \cdot c) \ , & a\urcorner \leq b\urcorner &\Rightarrow (a \cdot c)\urcorner \leq (b \cdot c)\urcorner \ . \end{aligned}\right\} \quad (8)$$

Our Kleene algebras LAN, REL and PAT are all typed. In REL we have, as usual,

$$\ulcorner R = R ; \top \cap I \ \wedge \ R\urcorner = \top ; R \cap I \ .$$

In LAN we have

$$\ulcorner U = U\urcorner = \begin{cases} \varepsilon \text{ if } U \neq \emptyset \ , \\ \emptyset \text{ otherwise.} \end{cases}$$

Finally, most relevant for our applications, in PAT we get

$$\ulcorner U = \mathit{first}(U) \ \wedge \ U\urcorner = \mathit{last}(U) \ ,$$

where *first* and *last* are the pointwise extensions of the operations given by

$$\begin{aligned} \mathit{first}(\varepsilon) &\stackrel{\text{def}}{=} \mathit{last}(\varepsilon) \stackrel{\text{def}}{=} \varepsilon \ , \\ \mathit{first}(a \bullet s) &\stackrel{\text{def}}{=} a \ , \qquad \mathit{last}(s \bullet a) \stackrel{\text{def}}{=} a \ , \end{aligned}$$

for $a \in A, s \in A^*$.

3.4 Truth Values and Assertions

The elements 1 and 0 of a Kleene algebra can play the roles of the truth values "true" and "false". Expressions that yield one of these values are therefore also called *assertions* (see e.g. [Möller 96]). The assertion 0 means not only "false", but also "undefined".

Negation is defined by

$$\neg 0 \stackrel{\text{def}}{=} 1 \ , \qquad \neg 1 \stackrel{\text{def}}{=} 0 \ .$$

Then for an assertion b and an element c we have

$$b \cdot c = c \cdot b = \begin{cases} c \text{ if } b = 1 \ , \\ 0 \text{ if } b = 0 \ . \end{cases}$$

In the sequel, for emphasis we shall always use the generic $\cdot$ when connecting assertions with expressions, regardless of the concrete Kleene algebra we are working in.

Note that 0 and 1 are types. The conjunction of types and hence assertions a, b is their infimum $a \sqcap b$ or, equivalently, their product $a \cdot b$; their disjunction is their sum $a + b$. We write $a \wedge b$ for $a \sqcap b$ and $a \vee b$ for $a + b$.

Using assertions we can construct a conditional and a guarded expression:

$$\begin{aligned} &\mathsf{if}\ b\,\mathsf{then}\,c\,\mathsf{else}\,d\ \mathsf{fi} \stackrel{\text{def}}{=} b \cdot c \ + \ \neg b \cdot d \ , \\ &\mathsf{if}\ b_1\,\mathsf{then}\,c_1 \ [] \ \cdots \ [] \ b_n\,\mathsf{then}\,c_n\ \mathsf{fi} \stackrel{\text{def}}{=} \sum_{i=1}^{n} b_i \cdot c_i \ . \end{aligned}$$

for assertions b, b_i and elements c, c_i, d. Note that the conditional is monotonic only in c and d. So recursions over the condition b need not be well-defined. A property we are going to use in the sequel is

$$\mathsf{if}\ b\,\mathsf{then}\,d\,\mathsf{else}\,\mathsf{if}\ c\,\mathsf{then}\,d\,\mathsf{else}\,e\ \mathsf{fi}\ \mathsf{fi} = \mathsf{if}\ b \vee c\,\mathsf{then}\,d\,\mathsf{else}\,e\ \mathsf{fi} \tag{9}$$

for assertions b, c and elements d, e.

In connection with recursions, assertions play the role of invariants as known from imperative programming. See [Möller 96] for rules for strengthening and weakening such invariants; invariant introduction and elimination are, of course, particular cases.

3.5 Filters

A particular case of assertions in the Kleene algebra LAN are filters. Given a parameterized assertion $B : A^* \to \{\emptyset, \varepsilon\}$, the filter $B \triangleleft$ is defined on words by

$$B \triangleleft w = B(w) \bullet w = \begin{cases} w \text{ if } B(w) = \varepsilon \ , \\ \emptyset \text{ if } B(w) = \emptyset \ . \end{cases}$$

The pointwise extension $B \triangleleft W$ of the filter to a language W yields those elements of W that satisfy B:

$$B \triangleleft W = \{w : w \in W \wedge B(w)\} \ .$$

This operation distributes through $\cap$ and $\cup$:

$$\begin{aligned} B \triangleleft (V \cap W) &= B \triangleleft V \cap B \triangleleft W \ , \\ B \triangleleft (V \cup W) &= B \triangleleft V \cup B \triangleleft W \ . \end{aligned}$$

By *filter promotion* we mean an application of the laws

$$\left.\begin{aligned} \bigcup_{x \in U} B(x) \cdot E(x) &= \bigcup_{x \in B \triangleleft U} E(x) \ , \\ \bigcup_{x \in U} F(x \cap V) &= \bigcup_{x \in U \cap V} F(x) \ , \end{aligned}\right\} \tag{10}$$

provided $F(\emptyset) = \emptyset$.

4 A General Graph Problem Class

Consider now a graph over node set A. We define a general graph processing operation E by

$$E(f, g)(W) \stackrel{\text{def}}{=} g(f(W)) \ , \tag{11}$$

where

- $W \subseteq A^*$ is a subset depending on the structure of the particular application. It might, e.g., be the set of all finite paths in the graph.
- $f : A^* \to M$ is an *abstraction* function from words to a "valuation" set M which might e.g. be the set of natural numbers if we are interested in counting edges in a path. f is extended pointwise to sets of words and hence is distributive, monotonic and strict.
- $g : \mathcal{P}(M) \to \mathcal{P}(M)$ is a *selection* function. It acts globally on $f(W)$, e.g., selects the minimum in the case of sets of natural numbers, and hence *is not* defined as a pointwise extension. Rather we assume the properties

  (GEN1) $g(K) \subseteq K$,
  (GEN2) $g(K \cup L) = g(g(K) \cup g(L))$ (weak distributivity),

  for $K, L \subseteq M$.

Weak distributivity means that g may be applied to subsets without changing the overall result. Note that (GEN2) implies idempotence of g and (GEN2) is equivalent to

(GEN2') $g(K \cup L) = g(K \cup g(L))$.

If g is a filter ($B \triangleleft$), then (GEN1) and (GEN2) hold automatically. The advantage of defining g not generally as a filter gives us the flexibility of admitting

non-distributive operations like the minimum. This difference in algebraic properties explains why we use two separate functions f and g rather than their combination into a single one.

The general operation E comprises quite a diversity of graph problems, embodied by different choices of W and additional constraints on f and g.

But we now abstract even further, moving away from the particular case of graphs. Assume that $(S, \Sigma, \cdot, 0, 1)$ is a typed Kleene algebra. We define the general operation E by

$$E(f,g)(w) \stackrel{\text{def}}{=} g(f(w)) \tag{12}$$

where

- $w \in S$ is a fixed element of S,
- $f : S \to \mathcal{P}(M)$ is a disjunctive *abstraction* function with some set M of "valuations", where a function f from a Kleene algebra into an upper semilattice is *disjunctive* if it distributes through $+$, i.e., satisfies $f(x+y) = f(x) \sqcup f(y)$,
- $g : \mathcal{P}(M) \to \mathcal{P}(M)$ is a *selection* satisfying the properties

  (GEN1) $g(K) \subseteq K$,
  (GEN2) $g(K \cup L) = g(g(K) \cup g(L))$ (weak distributivity),

  for $K, L \subseteq M$.

5 Layer Oriented Graph Traversals

A number of problems use the set of all paths starting in a set S and ending in a set T of nodes.

For some of these problems we define in this section a class, derive a basic algorithm and apply it to examples. At the end of this section we show how a more efficient version of the basic algorithm can be developed.

5.1 Definition

As mentioned above, we choose for W the set of all paths that start in end points of S and end in starting points of T, i.e., we set in (12) $W = S \bowtie R^{\rightsquigarrow} \bowtie T$ and specify a graph traversal operation F by

$$F(f,g)(S,R,T) \stackrel{\text{def}}{=} E(f,g)(S \bowtie R^{\rightsquigarrow} \bowtie T) = g(f(S \bowtie R^{\rightsquigarrow} \bowtie T)) \ ,$$

with $S, T \subseteq A^*$ and $R \subseteq A \bullet A$. Note that, contrary to [Russling 96b] we do not choose S and T as subsets of A. This eases the recursion step for the basic algorithm, as is shown in the next section.

We replace this graph theoretic formulation by one for general typed Kleene algebras. Here the definition of F reads

$$F(f,g)(a,b,c) \stackrel{\text{def}}{=} E(f,g)(a \cdot b^* \cdot c) = g(f(a \cdot b^* \cdot c)) \ ,$$

with $a, b, c \in S$.

5.2 Derivation of the Basic Algorithm

We now want to find a recursion equation for F. We calculate

$$
\begin{aligned}
& F(f,g)(a,b,c) \\
= & \quad \{\![\text{ definition }]\!\} \\
& g(f(a \cdot b^* \cdot c)) \\
= & \quad \{\![\text{ idempotence of } \cup \ ]\!\} \\
& g(f(a \cdot b^* \cdot c) \cup f(a \cdot b^* \cdot c)) \\
= & \quad \{\![\ (5) \ ]\!\} \\
& g(f(a \cdot (1 + b \cdot b^*) \cdot c) \cup f(a \cdot b^* \cdot c)) \\
= & \quad \{\![\text{ distributivity and neutrality }]\!\} \\
& g(f(a \cdot c + a \cdot b \cdot b^* \cdot c) \cup f(a \cdot b^* \cdot c)) \\
= & \quad \{\![\text{ associativity of } + \text{ and disjunctivity of } f \text{ twice }]\!\} \\
& g(f(a \cdot c) \cup f(a \cdot b \cdot b^* \cdot c + a \cdot b^* \cdot c)) \\
= & \quad \{\![\text{ distributivity }]\!\} \\
& g(f(a \cdot c) \cup f((a \cdot b + a) \cdot b^* \cdot c)) \\
= & \quad \{\![\text{ (GEN2') and commutativity of } + \ ]\!\} \\
& g(f(a \cdot c) \cup g(f((a + a \cdot b) \cdot b^* \cdot c))) \\
= & \quad \{\![\text{ definition }]\!\} \\
& g(f(a \cdot c) \cup F(f,g)(a + a \cdot b,\, b,\, c)) \ .
\end{aligned}
$$

5.3 Termination Cases

We prepare the introduction of termination cases by deriving another form of F:

$$
\begin{aligned}
& F(f,g)(a,b,c) \\
= & \quad \{\![\text{ definition }]\!\} \\
& g(f(a \cdot b^* \cdot c)) \\
= & \quad \{\![\text{ by (5) we have } b^* = 1 + b^* \ ]\!\} \\
& g(f(a \cdot (1 + b^*) \cdot c)) \\
= & \quad \{\![\text{ distributivity and neutrality }]\!\} \\
& g(f(a \cdot c + a \cdot b^* \cdot c)) \\
= & \quad \{\![\text{ disjunctivity of } f \ ]\!\} \\
& g(f(a \cdot c) \cup f(a \cdot b^* \cdot c)) \ . \qquad (13)
\end{aligned}
$$

Motivated by the graph theoretical applications we now postulate some conditions about f and g:

$$\begin{array}{ll} \text{(LAY1)} & \lceil c \ \leq \ a \rceil \ \Rightarrow \ g(f(a \cdot c) \cup f(a \cdot u \cdot c)) = g(f(a \cdot c)) \ , \\ \text{(LAY2)} & (a \cdot u)\rceil \ \leq \ a \rceil \ \Rightarrow \ g(f(a \cdot c) \cup f(a \cdot u \cdot c)) = g(f(a \cdot c)) \ , \end{array}$$

with $a, c, u \in S$. When dealing with graph problems, $\lceil c \leq a \rceil$ is the case where the set of starting nodes of c already is contained in the set of end nodes of a. The condition $(a \cdot u)\rceil \leq a\rceil$ means that by further traversal of the graph along u no new end nodes are reached. In both cases the second use of the abstraction f should give no new information and be ignored by g.
Case 1: $\lceil c \leq a \rceil$. Then we get by (LAY1) that

$$(13) \ = \ g(f(a \cdot c)) \ .$$

For the second case we need the following lemma:

Lemma 51 *For $a, b \in S$ one has $(a \cdot b)\rceil \ \leq \ a\rceil \ \Rightarrow \ (a \cdot b^*)\rceil \ \leq \ a\rceil$.*

Proof. We apply Lemma 1 (Closure Induction) of [Möller 93] with continuous predicate $P[x] \stackrel{\text{def}}{\Leftrightarrow} (a \cdot x)\rceil \leq a\rceil$ and $z = b$. Then we need to show $\forall \ c : P[c] \ \Rightarrow \ P[b \cdot c]$:

$$\begin{array}{ll} & (a \cdot (b \cdot c))\rceil \\ = & \quad \{\![\text{ associativity }]\!\} \\ & ((a \cdot b) \cdot c)\rceil \\ \leq & \quad \{\![\text{ assumption and (8) }]\!\} \\ & (a \cdot c)\rceil \\ \leq & \quad \{\![\ P[c] \ ]\!\} \\ & a\rceil \end{array}$$

From this and $P[1] \ \Leftrightarrow \ (a \cdot 1)\rceil \leq a\rceil$ we obtain the result. □

Now we can proceed with
Case 2: $(a \cdot b)\rceil \leq a\rceil$. Then by Lemma 51 and (LAY2) we get again

$$(13) \ = \ g(f(a \cdot c)) \ .$$

In sum we have derived the following basic algorithm:

$$\begin{array}{l} F(f,g)(a,b,c) = \mathsf{if} \ \lceil c \ \leq \ a\rceil \ \vee \ (a \cdot b)\rceil \ \leq \ a\rceil \\ \qquad\qquad\qquad\qquad \mathsf{then} \, g(f(a \cdot c)) \\ \qquad\qquad\qquad\qquad \mathsf{else} \ g(f(a \cdot c) \ \cup \ F(f,g)(a + a \cdot b, \, b, \, c)) \ \mathsf{fi} \ . \end{array} \tag{14}$$

This terminates whenever the set of types in the underlying Kleene algebra is upward noetherian, i.e, has no infinite $\leq$-ascending chains. This is always the case in LAN, whereas in REL and PAT it holds only if A is finite. A suitable termination measure is $a^\urcorner$, since

$$(a + a \cdot b)^\urcorner = a^\urcorner + (a \cdot b)^\urcorner \geq a^\urcorner$$

and

$$(a + a \cdot b)^\urcorner = a^\urcorner \Leftrightarrow (a \cdot b)^\urcorner \leq a^\urcorner \ .$$

5.4 Applications

Reachability Our first example is the problem to compute, given an edge set $R \subseteq A \bullet A$ and a set $S \subseteq A$ of nodes the set of all nodes which are reachable from S. We first give a solution in the Kleene algebra PAT and choose $f \stackrel{\text{def}}{=} \_^\urcorner$, $g \stackrel{\text{def}}{=} id$, $c \stackrel{\text{def}}{=} A$ and define, for $S \subseteq A^+$,

$$reach(S) \stackrel{\text{def}}{=} F(\_^\urcorner, id)(S, R, A) \ .$$

It is easily checked that $\_^\urcorner$ and id satisfy the required conditions.

By our definitions we can now solve the reachability problem recursively:

$$\begin{array}{ll}
 & reach(S) \\
= & \quad \{\![\text{ definition }]\!\} \\
 & F(_^\urcorner, id)(S, R, A) \\
= & \quad \{\![\text{ (14) }]\!\} \\
 & \mathsf{if}\ \ulcorner A \subseteq S^\urcorner \vee (S \bowtie R)^\urcorner \subseteq S^\urcorner \\
 & \quad \mathsf{then}\ (S \bowtie A)^\urcorner \\
 & \quad \mathsf{else}\ (S \bowtie A)^\urcorner \cup reach(S \cup S \bowtie R)\ \mathsf{fi} \\
= & \quad \{\![\text{ (1),(8) }]\!\} \\
 & \mathsf{if}\ A \subseteq S^\urcorner \vee (S \bowtie R)^\urcorner \subseteq S^\urcorner \\
 & \quad \mathsf{then}\ S^\urcorner \\
 & \quad \mathsf{else}\ S^\urcorner \cup reach(S \cup S \bowtie R)\ \mathsf{fi} \\
= & \quad \{\![\ A \subseteq S^\urcorner \Rightarrow A = S^\urcorner \Rightarrow (S \bowtie R)^\urcorner \subseteq S^\urcorner \ ]\!\} \\
 & \mathsf{if}\ (S \bowtie R)^\urcorner \subseteq S^\urcorner \\
 & \quad \mathsf{then}\ S^\urcorner \\
 & \quad \mathsf{else}\ S^\urcorner \cup reach(S \cup S \bowtie R)\ \mathsf{fi} \ .
\end{array}$$

Alternatively, we can solve the reachability problem in REL by setting, for $Q \subseteq A$

$$relreach(Q) \stackrel{\text{def}}{=} F(\_^\urcorner, id)(I_Q, R, I_A) \ .$$

Again the required conditions are easily shown. The resulting algorithm is

$$
\begin{array}{l}
relreach(Q) \;=\; rr(I_Q)\ , \\
rr(S) \;= \\
\qquad \mathsf{if}\ (S\,;R)^{\urcorner} \subseteq S^{\urcorner} \\
\qquad\quad \mathsf{then}\ S^{\urcorner} \\
\qquad\quad \mathsf{else}\ S^{\urcorner} \cup rr(S \cup S\,;R)\ \mathsf{fi}\ .
\end{array}
$$

Shortest Connecting Path We define, in PAT,

$$
shortestpaths(S,T) \stackrel{\text{def}}{=} F(id, minpaths)(S,R,T)\ ,
$$

with

$$
\begin{array}{l}
minpaths(U) \stackrel{\text{def}}{=} \mathsf{let}\ ml \;=\; min(\|U\|)\ \mathsf{in}\ \ lg(ml) \triangleleft U\ , \\
lg(n)(x) \;=\; (\|x\| = n)\ .
\end{array}
$$

Here we use the pointwise extension of $\|\_\|$ to languages. Hence *minpaths* selects from a set of words the ones with the least number of letters. Again the conditions (GEN) and (LAY1,LAY2) are satisfied. Therefore we have the following algorithm for computing the shortest path between a set S and the node y:

$$
\begin{array}{ll}
 & shortestpaths(S,y) \\
= & \quad \{\![\ \text{definition}\ ]\!\} \\
 & F(id, minpaths)(S,R,y) \\
= & \quad \{\![\ (14)\ ]\!\} \\
 & \mathsf{if}\ \ulcorner y \subseteq S^{\urcorner} \vee (S \bowtie R)^{\urcorner} \subseteq S^{\urcorner} \\
 & \quad \mathsf{then}\ minpaths(S \bowtie y) \\
 & \quad \mathsf{else}\ \ minpaths(S \bowtie y \cup shortestpaths(S \cup S \bowtie R, y))\ \mathsf{fi} \\
= & \quad \{\![\ \text{set theory, } (9)\ ]\!\} \\
 & \mathsf{if}\ y \in S^{\urcorner} \\
 & \quad \mathsf{then}\ minpaths(S \bowtie y) \\
 & \quad \mathsf{else}\ \mathsf{if}\ (S \bowtie R)^{\urcorner} \subseteq S^{\urcorner} \\
 & \qquad\quad \mathsf{then}\ minpaths(S \bowtie y) \\
 & \qquad\quad \mathsf{else}\ \ minpaths(S \bowtie y \cup shortestpaths(S \cup S \bowtie R, y))\ \mathsf{fi}\ \mathsf{fi} \\
= & \quad \{\![\ y \notin S^{\urcorner} \Rightarrow S \bowtie y = \emptyset\ ]\!\} \\
 & \mathsf{if}\ y \in S^{\urcorner} \\
 & \quad \mathsf{then}\ minpaths(S \bowtie y) \\
 & \quad \mathsf{else}\ \mathsf{if}\ (S \bowtie R)^{\urcorner} \subseteq S^{\urcorner} \\
 & \qquad\quad \mathsf{then}\ minpaths(\emptyset) \\
 & \qquad\quad \mathsf{else}\ \ minpaths(shortestpaths(S \cup S \bowtie R, y))\ \mathsf{fi}\ \mathsf{fi}\ .
\end{array}
$$

Since *minpaths* is defined via a filter, we have

$$
minpaths(\emptyset) \;=\; \emptyset\ .
$$

We now simplify the second else-branch.

$$
\begin{array}{ll}
& \mathit{minpaths}(\mathit{shortestpaths}(S \cup S \bowtie R, y)) \\
= & \quad \{\![\text{ definition of } \mathit{shortestpaths} \ ]\!\} \\
& \mathit{minpaths}(F(\mathit{id}, \mathit{minpaths})(S \cup S \bowtie R, R, y)) \\
= & \quad \{\![\text{ definition of } F \ ]\!\} \\
& \mathit{minpaths}(\mathit{minpaths}(\mathit{id}(S \cup S \bowtie R, y))) \\
= & \quad \{\![\text{ idempotence of } \mathit{minpaths} \ ]\!\} \\
& \mathit{minpaths}(\mathit{id}(S \cup S \bowtie R, y)) \\
= & \quad \{\![\text{ definition of } F \ ]\!\} \\
& F(\mathit{id}, \mathit{minpaths})(S \cup S \bowtie R, R, y) \\
= & \quad \{\![\text{ definition of } \mathit{shortestpaths} \ ]\!\} \\
& \mathit{shortestpaths}(S \cup S \bowtie R, y) \ .
\end{array}
$$

Altogether,

$$
\begin{array}{l}
\mathit{shortestpaths}(S, y) = \\
\quad \mathsf{if}\ y \in S^\neg \\
\quad\quad \mathsf{then}\ \mathit{minpaths}(S \bowtie y) \\
\quad\quad \mathsf{else}\ \mathsf{if}\ (S \bowtie R)^\neg \subseteq S^\neg \\
\quad\quad\quad\quad \mathsf{then}\ \emptyset \\
\quad\quad\quad\quad \mathsf{else}\ \ \mathit{shortestpaths}(S \cup S \bowtie R, y)\ \mathsf{fi}\ \mathsf{fi} \ .
\end{array}
$$

Note that, in view of the law $(a \cdot b)^\neg = (a^\neg \cdot b)^\neg$ in (8), we only need to carry along $S^\neg$ rather than all of S. Together with the efficiency improvement in the next subsection this brings the algorithm down to a complexity of $O(|A| + |R|)$.

5.5 Improved Efficiency

In graph applications, the parameter a in algorithm (14) carries all paths of the graph which have already been visited during the layered traversal from the starting set. We shall now improve the efficiency of the algorithm by introducing an additional parameter u that contains all already computed paths, while a carries only those paths whose last node has not been visited by any other path. This can again be done in the more general framework of typed Kleene algebras.

We define, using an assertion,

$$
F_{\mathit{eff}}(f, g)(a, b, c, u) = (a^\neg \sqcap u^\neg = 0) \cdot F(f, g)(a + u, b, c) \ .
$$

From this we get immediately

$$
F(f, g)(a, b, c) = F_{\mathit{eff}}(f, g)(a, b, c, 0) \ . \tag{15}
$$

Let now $v \stackrel{\text{def}}{=} a + u$ and assume $a^\urcorner \sqcap u^\urcorner = 0$. By (14) and (15) we get the following termination case:

$$\begin{array}{l} \ulcorner c \;\leq\; v^\urcorner \;\vee\; (v \cdot b)^\urcorner \;\leq\; v^\urcorner \Rightarrow \\ F_{\mathit{eff}}(f,g)(a,b,c,u) \;=\; g(f(v \cdot c)) \;\; . \end{array}$$

The recursive case looks as follows:

$$\begin{array}{ll} & F_{\mathit{eff}}(f,g)(a,b,c,u) \\ = & \quad \{\![\text{ definitions }]\!\} \\ & F(f,g)(v,b,c) \\ = & \quad \{\![\text{ (14) }]\!\} \\ & g(f(v \cdot c) + g(f((v + v \cdot b) \cdot b^* \cdot c))) \\ = & \quad \{\![\text{ let } b_1 = b \cdot v^\urcorner, b_2 = b \backslash b_1 \;\;]\!\} \\ & g(f(v \cdot b) + g(f((v + v \cdot (b_1 + b_2)) \cdot b^* \cdot c))) \\ = & \quad \{\![\text{ distributivity and disjunctivity of } f, \text{ commutativity} \\ & \quad \text{and associativity of } +, \text{ several times (GEN2') }]\!\} \\ & g(f(v \cdot b) + f(v \cdot b_2 \cdot b^* \cdot c) + g(f(v \cdot b^* \cdot c) + f(v \cdot b_1 \cdot b^* \cdot c))) \\ = & \quad \{\![\text{ definition of } b_1, \text{ (8), (LAY2) }]\!\} \\ & g(f(v \cdot b) + f(v \cdot b_2 \cdot b^* \cdot c) + g(f(v \cdot b^* \cdot c))) \\ = & \quad \{\![\text{ (GEN2') twice, commutativity of } + \text{ and distributivity }]\!\} \\ & g(f(v \cdot b) + g(f((v + v \cdot b_2) \cdot b^* \cdot c))) \\ = & \quad \{\![\text{ definition of } F \;\;]\!\} \\ & g(f(v \cdot b) + F(f,g)(v + v \cdot b_2, b, c)) \\ = & \quad \{\![\text{ definition of } F_{\mathit{eff}} \;\;]\!\} \\ & g(f(v \cdot b) + F_{\mathit{eff}}(f,g)(v \cdot b_2, b, c, v)) \;\; . \end{array}$$

It is easy to see that the recursive call preserves the invariant. In sum we get

$$\begin{array}{l} F_{\mathit{eff}}(f,g)(a,b,c,u) = \\ \quad \mathsf{let}\ v \;=\; a + u \\ \qquad\; b_1 \;=\; b \cdot v^\urcorner \\ \qquad\; b_2 \;=\; b \backslash b_1 \\ \quad \mathsf{in}\ \; (a^\urcorner \sqcap u^\urcorner = 0) \cdot \\ \qquad \mathsf{if}\ \ulcorner c \;\leq\; v^\urcorner \;\vee\; (v \cdot b)^\urcorner \;\leq\; v^\urcorner \\ \qquad\quad \mathsf{then}\ g(f(v \cdot c)) \\ \qquad\quad \mathsf{else}\ \; g(f(v \cdot c) + F_{\mathit{eff}}(f,g)(v \cdot b_2, b, c, v))\ \mathsf{fi} \;\; . \end{array} \tag{16}$$

One checks easily that we can strengthen the invariant by the conjunct $u \cdot b_2 \leq a + u$. With its help, we can simplify the algorithm to

$$
\begin{array}{l}
F_{\mathit{eff2}}(f,g)(a,b,c,u) = \\
\quad \mathsf{let}\ v\ =\ a+u \\
\qquad b_1\ =\ b\cdot v^{\urcorner} \\
\qquad b_2\ =\ b\backslash b_1 \\
\quad \mathsf{in}\ \ (a^{\urcorner}\sqcap u^{\urcorner}=0\ \wedge\ u\cdot b_2\leq v)\cdot \\
\qquad \mathsf{if}\ {\ulcorner}c\ \leq\ v^{\urcorner}\ \ \vee\ \ (a\cdot b_2)^{\urcorner}\ \leq\ v^{\urcorner} \\
\qquad\quad \mathsf{then}\, g(f(v\cdot c)) \\
\qquad\quad \mathsf{else}\ \ g(f(v\cdot c)+F_{\mathit{eff2}}(f,g)(a\cdot b_2,b,c,v))\ \mathsf{fi}\ \ .
\end{array}
\tag{17}
$$

Here the expensive computation of $v \cdot b_2$ is reduced to that of $a \cdot b_2$.

6 Hamiltonian Path Problems

In this section we define a class of graph problems for the case where the set of nodes is ordered. An order can be given by a binary relation between the nodes or by a permutation of the node set. Since permutations are Hamiltonian paths, the latter will serve as the basis of our further considerations.

6.1 Definition

In 1859 Sir William Hamilton suggested the game "around the world ": the points of a dodecahedron were named after cities and the task of the game was to plan a round trip along the edges of the dodecahedron such that each city was visited exactly once.

A *Hamiltonian path* for an alphabet A and a binary relation $R \subseteq A \bullet A$ is a path which traverses each node of the respective graph exactly once. Hence the set of Hamiltonian paths is defined as follows:

$$
\mathit{hamiltonianpaths} \stackrel{\mathrm{def}}{=} \mathit{perms}(A) \cap R^{\rightsquigarrow}\ \ ,
$$

with

$$
\begin{array}{l}
\mathit{perms}(\emptyset) \stackrel{\mathrm{def}}{=} \varepsilon\ \ , \\
\mathit{perms}(S) \stackrel{\mathrm{def}}{=} \bigcup\limits_{x\in S} x \bullet \mathit{perms}(S\backslash x)
\end{array}
$$

for $S \neq \emptyset$. This set of Hamiltonian paths is, as mentioned above, the underlying language of the instantiation of our general operation we want to specify now. As selection function we use in this section a filter operation $B\triangleleft$. In the sequel, when a recursive equation for *hamiltonianpaths* is available, we do not want to apply the test B only to the paths in the end result. This would be inefficient, since in this way we would compute many paths which in the end would be filtered out again by B. To avoid this we require that B be suffix closed, a property which is defined as follows:

An assertion $B(u)$ is *suffix closed* for a language $U \subseteq A^*$ iff for all $v \bullet w \in U$ with $\|w\| \geq 1$ one has

$$
B(v \bullet w) \leq B(w)\ \ , \tag{18}
$$

i.e., $B(v \bullet w)$ implies $B(w)$. If then B doesn't hold for a non-empty suffix of a repetition free path then it doesn't hold for the complete path either. A path or a word is *repetition free* if no node or no letter appears twice or more in it.

Therefore we choose in (11) $W \stackrel{\text{def}}{=} perms(A) \cap R^{\rightsquigarrow}$, $f \stackrel{\text{def}}{=} id$ and $g \stackrel{\text{def}}{=} B \triangleleft$ and obtain in this way the Hamiltonian path operation H as

$$H(B) \stackrel{\text{def}}{=} E(perms(A) \cap R^{\rightsquigarrow})(id, B) = B \triangleleft hamiltonianpaths \ ,$$

with the condition that

(HAM) B is a suffix closed assertion for the set of all repetition free paths.

6.2 Derivation of the Basic Algorithm

To derive a basic algorithm for this problem class we need a recursive version of *hamiltonianpaths*. To this end we generalize *hamiltonianpaths* in the following way:

$$hamiltonianpaths = hp(|A|) \ .$$

hp computes the set of repetition free paths of length n:

$$hp(n) \stackrel{\text{def}}{=} partperms(n) \cap R^{\rightsquigarrow} \ ,$$

where $partperms(n)$ is the set of partial permutations of length n:

$$partperms(n) \stackrel{\text{def}}{=} \bigcup_{T \subseteq A \wedge |T|=n} perms(T) \ .$$

One sees easily that $n > |A|$ implies $partperms(n) = \emptyset$ and hence

$$hp(n) = \emptyset \ .$$

Moreover, $n \leq |A|$ implies $\mathsf{ar}\, partperms(n) = n$ and hence, for $n \geq 1$,

$$hp(n) = partperms(n) \cap {}^{n-1}R \ . \tag{19}$$

In the sequel we denote the set of letters occurring in a word p by $set(p)$. we omit the straightforward inductive definition of *set* but note that it is extended pointwise to languages. Define now, for $U \subseteq A^*$,

$$non(U) \stackrel{\text{def}}{=} A \backslash set(U) \ .$$

For the transformation into recursive form we need an auxiliary lemma:

Lemma 61 *For $n \neq 0$ one has*

$$partperms(n) = \bigcup_{T \subseteq A \wedge |T|=n-1} non(T) \bullet perms(T) \ .$$

Proof. see [Russling 96b]. □

Now we can perform a case distinction for *partperms*.
Case 1: $n = 0$. From the definitions we obtain immediately

$$partperms(0) = \varepsilon \ .$$

Case 2: $n \geq 1$.

$$\begin{aligned}
& partperms(n) \\
= \quad & \{\![\text{ Lemma 61 }]\!\} \\
& \bigcup_{T \subseteq A \wedge |T| = n-1} non(T) \bullet perms(T) \\
= \quad & \{\![\text{ pointwise extension and } p \in perms(T) \Rightarrow set(p) = T \ ]\!\} \\
& \bigcup_{T \subseteq A \wedge |T| = n-1} \ \bigcup_{p \in perms(T)} non(p) \bullet p \\
= \quad & \{\![\text{ definition of } partperms \ ]\!\} \\
& \bigcup_{p \in partperms(n-1)} non(p) \bullet p \ .
\end{aligned}$$

We extend the case distinction to *hp*.
Case 1: $n = 0$. From the definitions we obtain immediately

$$hp(0) = \varepsilon \ .$$

Case 2: $n = 1$. From (19) we get

$$hp(1) = A \ .$$

For the next case we need

Lemma 62 *For $S \subseteq A$ and $R \subseteq A \bullet A$ one has:*

$$S \bullet U \cap R \bowtie V = S \bowtie R \bowtie (U \cap V).$$

Proof. see [Russling 96a]. □

Case 3: $n > 1$.

$$\begin{aligned}
& hp(n) \\
= \quad & \{\![\text{ (19) }]\!\} \\
& partperms(n) \cap {}^{n-1}R \\
= \quad & \{\![\text{ case distinction of } partperms \text{ and (7) }]\!\} \\
& \Big(\bigcup_{p \in partperms(n-1)} non(p) \bullet p \Big) \cap R \bowtie {}^{n-2}R
\end{aligned}$$

$$
\begin{aligned}
&= \quad \{\![\text{ distributivity }]\!\} \\
&\quad \bigcup_{p \in partperms(n-1)} non(p) \bullet p \ \cap \ R \bowtie {}^{n-2}R \\
&= \quad \{\![\text{ Lemma 62 }]\!\} \\
&\quad \bigcup_{p \in partperms(n-1)} non(p) \bowtie R \bowtie (p \cap {}^{n-2}R) \\
&= \quad \{\![\text{ filter promotion (10) }]\!\} \\
&\quad \bigcup_{p \in partperms(n-1) \cap {}^{n-2}R} non(p) \bowtie R \bowtie p \\
&= \quad \{\![\text{ (19) }]\!\} \\
&\quad \bigcup_{p \in hp(n-1)} non(p) \bowtie R \bowtie p.
\end{aligned}
$$

In sum we get for hp:

$$
\begin{aligned}
hp(n) = {} & \mathsf{if}\ n = 0\ \mathsf{then}\ \varepsilon \\
& [\!]\ n = 1\ \mathsf{then}\ A \\
& [\!]\ n > 1\ \mathsf{then} \bigcup_{p \in hp(n-1)} non(p) \bowtie R \bowtie p\ \mathsf{fi}.
\end{aligned} \tag{20}
$$

Termination is guaranteed by the decreasing parameter n.

Analogously to *hamiltonianpaths* we generalize the Hamiltonian path operation H to an operation HP which computes the set of repetition free paths of length n that satisfy B:

$$HP(B)(n) \stackrel{\text{def}}{=} B \lhd hp(n) \ . \tag{21}$$

Because of

$$H(B) = HP(B)(|A|) \tag{22}$$

we perform also for HP a case distinction.

Case 1: $n = 0$. From the definition of HP and (20) we get immediately

$$HP(B)(0) \ = \ B \lhd \varepsilon \ .$$

Case 2: $n = 1$. Again, the definition and (20) yield

$$HP(B)(1) \ = \ B \lhd A \ .$$

Case 3: $n > 1$.

$$
\begin{aligned}
& HP(B)(n) \\
= \quad & \{\![\text{ definition }]\!\} \\
& B \lhd hp(n)
\end{aligned}
$$

$$
\begin{aligned}
= \quad & \{\![\text{ (20) }]\!\} \\
& B \lhd \bigcup_{p \in hp(n-1)} non(p) \bowtie R \bowtie p \\
= \quad & \{\![\text{ distributivity }]\!\} \\
& \bigcup_{p \in hp(n-1)} B \lhd (non(p) \bowtie R \bowtie p) \\
= \quad & \{\![\text{ (HAM) B is suffix closed } \Rightarrow \\
& \qquad \text{for } q \in \ non(p) \bowtie R \bowtie p : B(q) \leq B(p) \ ]\!\} \\
& \bigcup_{p \in hp(n-1)} (B \lhd (non(p) \bowtie R \bowtie p)) \cdot B(p) \\
= \quad & \{\![\text{ filter promotion (10) }]\!\} \\
& \bigcup_{p \in B \lhd hp(n-1)} B \lhd (non(p) \bowtie R \bowtie p) \\
= \quad & \{\![\text{ definition }]\!\} \\
& \bigcup_{p \in HP(n-1)} B \lhd (non(p) \bowtie R \bowtie p) \ .
\end{aligned}
$$

In sum we have

$$
\begin{aligned}
HP(B)(n) = \ & \mathsf{if}\ n = 0\ \mathsf{then}\ B \lhd \varepsilon \\
& [\!]\ n = 1\ \mathsf{then}\ B \lhd A \\
& [\!]\ n > 1\ \mathsf{then} \bigcup_{p \in HP(B)(n-1)} B \lhd (non(p) \bowtie R \bowtie p)\ \mathsf{fi}\ .
\end{aligned} \tag{23}
$$

For $A \neq \emptyset$ the algorithm starts with all paths of length 1, i.e., the nodes that satisfy B. It then repeatedly attaches to the front ends of the already obtained paths those nodes which have not yet been used and still maintain B. Because of its way of traversing the graph in which every visit of a node may lead to many others the algorithm is also known as a *hydramorphism* after the hydra from Greek Mythology.

A derivation of a more efficient version of the basic algorithm can be found in [Russling 96b].

6.3 Applications

Topological Sorting We now want to study the problem of topologically sorting a set A w.r.t. a relation $Q \subseteq A \bullet A$. A *topological sorting* of A is a permutation of the A such that for arbitrary $a, b \in A$ one has: if $a \bullet b \in Q$ then a must occur in the sorting before b.

We first want to attack the problem from the rear and compute the set of all relations which are admissible for a given sorting $s \in perms(A)$: The first letter of s may be in relation Q with any other letter, the next one with all those

which occur after it and so on. So we can define the greatest admissible relation inductively by

$$\begin{aligned} \mathit{allowedrel}(\varepsilon) &\stackrel{\text{def}}{=} \emptyset, \\ \mathit{allowedrel}(a \bullet w) &\stackrel{\text{def}}{=} a \bullet \mathit{set}(w) \cup \mathit{allowedrel}(w) \ , \end{aligned}$$

with $a \in A$ and $w \in A^*$ so that $a \bullet w$ is repetition free. A permutation $w \in \mathit{perms}(A)$ is therefore a topological sorting w.r.t. Q iff $Q \subseteq \mathit{allowedrel}(w)$.

We are now in the position to solve the problem by application of the Hamiltonian path operation choosing $R \stackrel{\text{def}}{=} A \bullet A$, i.e., first admitting all permutations and then incorporating Q into a filter operation to check $Q \subseteq \mathit{allowedrel}(w)$.

This assertion is, however, not suffix closed, because it is not applicable to partial permutations. Hence we must extend it to the assertion $\mathit{consistent}(Q)(w)$ which checks Q also for a suffix of a topological sorting as admissible relation:

$$\mathit{consistent}(Q)(w) \stackrel{\text{def}}{=} Q \subseteq \mathit{allowedrel}(w) \cup \mathit{non}(w) \bullet A \ .$$

This means that no restriction is placed on points outside $\mathit{set}(w)$. Note that for $w \in \mathit{perms}(A)$ one obviously has

$$\mathit{consistent}(Q)(w) = Q \subseteq \mathit{allowedrel}(w) \ .$$

For the following Lemma we introduce the set of sinks relative to relation Q and set $B \subseteq A$ by

$$\mathit{sinks}(Q)(B) \stackrel{\text{def}}{=} \{a \in B : Q \cap a \bullet B = \emptyset\} \ .$$

Lemma 63 *For $a \in A, t \in A^*$ and $a \bullet t \in \bigcup_{i \in \mathbb{N}} \mathit{hp}(i)$ we have*

$$\mathit{consistent}(Q)(a \bullet t) = \mathit{consistent}(Q)(t) \wedge a \in \mathit{sinks}(Q)(\mathit{non}(t)) \ .$$

In particular, $\mathit{consistent}(Q)$ is suffix-closed on $\bigcup_{i \in \mathbb{N}} \mathit{hp}(i)$.

Proof. We calculate,

$$\begin{aligned} & \mathit{consistent}(Q)(a \bullet t) \\ = \quad & \{\![\text{ definition of } \mathit{consistent} \ ]\!\} \\ & Q \subseteq \mathit{allowedrel}(a \bullet t) \cup \mathit{non}(a \bullet t) \bullet A \\ = \quad & \{\![\text{ definition of } \mathit{allowedrel} \text{ and set theory }]\!\} \\ & Q \subseteq \mathit{allowedrel}(t) \cup a \bullet \mathit{set}(t) \cup \mathit{non}(a \bullet t) \bullet A \\ = \quad & \{\![\text{ set theory }]\!\} \\ & Q \subseteq \mathit{allowedrel}(t) \cup a \bullet A \cup \mathit{non}(a \bullet t) \bullet A \ \wedge \\ & Q \cap a \bullet \mathit{non}(t) = \emptyset \end{aligned}$$

$$
\begin{array}{ll}
= & \{\![\text{ set theory, definition of } \mathit{non} \text{ and distributivity }]\!\} \\
& Q \subseteq \mathit{allowedrel}(t) \cup \mathit{non}(t) \bullet A \ \wedge \\
& Q \cap a \bullet \mathit{non}(t) = \emptyset \\
= & \{\![\text{ definition of } \mathit{consistent}, \text{ definition of } \mathit{sinks}, \\
& \quad \text{since } a \bullet t \text{ is repetition-free and hence } a \in \ \mathit{non}(t) \ ]\!\} \\
& \mathit{consistent}(Q)(t) \ \wedge \ a \in \mathit{sinks}(Q)(\mathit{non}(t)) \ .
\end{array}
$$

Thus $\mathit{consistent}(a \bullet t) \leq \mathit{consistent}(t)$. □

Now we specify, for $Q \subseteq A \bullet A$ and $R \stackrel{\text{def}}{=} A \bullet A$, the set of topological sortings of A w.r.t. Q by

$$
\mathit{topsort} \stackrel{\text{def}}{=} H(\mathit{consistent}(Q)) = \mathit{consistent}(Q) \triangleleft \mathit{perms}(A) \ .
$$

Using (22) we define, with $R = A \bullet A$,

$$
\mathit{conshp}(n) \stackrel{\text{def}}{=} \mathit{HP}(\mathit{consistent}(Q))(n) = \mathit{consistent}(Q) \triangleleft \mathit{hp}(n) \ .
$$

Hence we have the following intermediary result:

$$
\mathit{topsort} = \mathit{conshp}(|A|) \ ,
$$

$$
\begin{array}{lll}
\mathit{conshp}(n) = & \mathsf{if}\, n = 0\, \mathsf{then}\, \mathit{consistent}(Q) \triangleleft \varepsilon & \\
& [\!] \ n = 1\, \mathsf{then}\, \mathit{consistent}(Q) \triangleleft A & \\
& [\!] \ n > 1\, \mathsf{then} \bigcup_{p \in \mathit{conshp}(n-1)} \mathit{consistent}(Q) \triangleleft (\mathit{non}(p) \bowtie R \bowtie p)\ \mathsf{fi} \ . & (24)
\end{array}
$$

Case 1: $n = 0$. From the definition of *consistent* we get immediately

$$
\mathit{conshp}(0) \ = \ \varepsilon \ .
$$

Case 2: $n = 1$.

$$
\begin{array}{ll}
& a \in \mathit{conshp}(1) \\
= & \{\![\text{ by (24) and definition of } \mathit{consistent} \ ]\!\} \\
& Q \subseteq \mathit{allowedrel}(a) \cup \mathit{non}(a) \bullet A \\
= & \{\![\text{ definition of } \mathit{allowedrel} \ ]\!\} \\
& Q \subseteq \mathit{non}(a) \bullet A \\
= & \{\![\text{ set theory }]\!\} \\
& a \bullet A \cap Q = \emptyset \\
= & \{\![\text{ definition of } \mathit{sinks} \ ]\!\} \\
& a \in \mathit{sinks}(Q)(A) \ .
\end{array}
$$

Case 3: $n > 1$. We first calculate within (24)

$$\begin{aligned}
& non(p) \bowtie R \bowtie p \\
= \quad & \{\![\text{ by } R = A \bullet A \text{ and associativity (2) }]\!\} \\
& non(p) \bowtie A \bullet A \bowtie p \\
= \quad & \{\![\text{ neutrality twice }]\!\} \\
& non(p) \bullet p \ .
\end{aligned}$$

Now, for $p \in conshp(n-1)$ and $x \in non(p)$

$$\begin{aligned}
& consistent(Q)(x \bullet p) \\
= \quad & \{\![\text{ Lemma 63 }]\!\} \\
& consistent(Q)(p) \ \wedge \ x \in sinks(Q)(non(p)) \\
= \quad & \{\![\ p \in conshp(n-1) \ = \ consistent(Q) \triangleleft hp(n-1), \\
& \quad \text{hence } consistent(Q)(p) \ ]\!\} \\
& x \in sinks(Q)(non(p)) \ ,
\end{aligned}$$

so that by filter promotion (10) and pointwise extension we can in (24) reduce the expression for $n > 1$ to

$$\bigcup_{p \in conshp(n-1)} sinks(Q)(non(p)) \bullet p \ .$$

Altogether,

$$\begin{aligned}
conshp(n) = \ & \mathsf{if}\ n = 0\ \mathsf{then}\ \varepsilon \\
& [\!]\ n = 1\ \mathsf{then}\ sinks(Q)(A) \\
& [\!]\ n > 1\ \mathsf{then} \bigcup_{p \in conshp(n-1)} sinks(Q)(non(p)) \bullet p\ \mathsf{fi} \ .
\end{aligned} \tag{25}$$

This is the standard removal-of-sinks algorithm. It can be implemented in complexity $O(|A|+|Q|)$ using an array of adjacency lists and in-degrees together with a linked list of relative sinks. A formal treatment of the array of in-degrees can be found in [Möller, Russling 93] in connection with an algorithm for cycle detection.

The Maximal Matching Problem in Directed, Bipartite Graphs Assume an alphabet A and a binary relation $R \subseteq A \bullet A$. In the sequel we consider *bipartite* directed graphs. This means that there are subsets $U, V \subseteq A$ with $A = U \cup V$, $U \cap V = \emptyset$ and $R \subseteq U \bullet V$. One may wonder why we did not include the summand $V \bullet U$ as well. However, we will work with the symmetric closure of our relations anyway.

A matching is a subset of the edges with the property that each node of the graph may be at most once starting point or end point of an edge of the matching.

More precisely, $M \subseteq R$ is a *matching* if $\forall\, m \in M : \mathit{ends}(M \backslash m) \cap \mathit{ends}(m) = \emptyset$, where

$$\begin{aligned} \mathit{ends}(\varepsilon) &= \emptyset \\ \mathit{ends}(a) &= \{a\} \\ \mathit{ends}(a \bullet u \bullet b) &= \{a\} \cup \{b\}, \end{aligned}$$

for $a, b \in A, u \in A^*$. The function *ends* is extended pointwise to languages.

The *maximal matching problem* consists in finding matchings with maximal cardinality $|M|$.

In the sequel we do not derive an algorithm for this problem from scratch. Rather we show that one subalgorithm of the standard solution is another particular instance of the general Hamiltonian operation.

By $\overset{\leftrightarrow}{M}$ we now denote the symmetric closure of M. A path $x_0 \bullet x_1 \bullet \cdots \bullet x_n \in R^{\rightsquigarrow}$ of vertices is an *alternating chain* w.r.t. M if the pairs $x_i \bullet x_{i+1}$ alternatingly lie within and without of $\overset{\leftrightarrow}{M}$. Formally, the assertion $\mathit{alter}(M)(w)$ is inductively defined by

$$\begin{aligned} \mathit{alter}(M)(\varepsilon) &= 0 \ , \\ \mathit{alter}(M)(a) &= 1 \ , \\ \mathit{alter}(M)(a \bullet b) &= a \bullet b \in \overset{\leftrightarrow}{R} \\ \mathit{alter}(M)(a \bullet b \bullet c) &= (a \bullet b \in \overset{\leftrightarrow}{R} \setminus \overset{\leftrightarrow}{M} \ \wedge b \bullet c \in \overset{\leftrightarrow}{M}) \vee \\ &\quad\ (a \bullet b \in \overset{\leftrightarrow}{M} \ \wedge b \bullet c \in \overset{\leftrightarrow}{R} \setminus \overset{\leftrightarrow}{M}) \\ \mathit{alter}(M)(a \bullet b \bullet c \bullet u) &= \mathit{alter}(M)(a \bullet b \bullet c) \wedge \\ &\quad\ \mathit{alter}(M)(b \bullet c \bullet u) \ , \end{aligned}$$

for $a, b, c \in A$ and $u \in A^*$. By this inductive definition, $\mathit{alter}(M)$ is obviously suffix closed.

A node a is *isolated* if it is not touched by any edge in M, i.e., if $x \notin \mathit{ends}(M)$. Formally,

$$\mathit{isolated}(M) \stackrel{\text{def}}{=} A \backslash \mathit{ends}(M) \ .$$

An alternating chain is *increasing* if its extremal nodes x_0 and x_n are isolated. Note that the length of an increasing chain always is odd and the starting and end points do not lie both in U nor both in V. If there now exists an increasing chain for a matching M, then one can construct a larger matching by omitting all edges of the chain that are contained in the matching and adds those edges of the chain which were not yet contained in M. For this we use the function

$$\mathit{symdiff}\,(M, c) \stackrel{\text{def}}{=} \mathit{edges}(c) \backslash M \cup M \backslash \mathit{edges}(c) \tag{26}$$

where

$$\begin{aligned} \mathit{edges}(\varepsilon) &\stackrel{\text{def}}{=} \emptyset \stackrel{\text{def}}{=} \mathit{edges}(a) \ , \\ \mathit{edges}(a \bullet b \bullet c) &\stackrel{\text{def}}{=} a \bullet b \cup \mathit{edges}(b \bullet c) \ . \end{aligned}$$

The fact that using an increasing chain one can construct a larger matching leads to the following recursive approach. In an auxiliary function G we compute for an arbitrary matching the set of corresponding increasing chains (function:

calc_aac). If this set is empty the matching is maximal. Otherwise one computes a larger matching by *symdiff* and calls G with the new matching as argument. Additional arguments of G are the sets of isolated nodes of U and V; they allow an efficient computation of the increasing alternating chains. G is initially called with the empty set, U and V. The correctness of the algorithm is established by

Theorem 64 *A matching has maximal cardinality iff no increasing chain exists for it.*

For the proof see e.g. [Biggs 89]. Therefore we have the specification

$$maxmatch \stackrel{\text{def}}{=} G(\emptyset, U, V)$$

where

$$\begin{array}{l} G(M,U,V) \stackrel{\text{def}}{=} \\ \mathsf{let}\ aac = calc\_aac(M,U,V) \\ \mathsf{in}\ \ \mathsf{if}\ aac = \emptyset \\ \qquad \mathsf{then}\ M \\ \qquad \mathsf{else} \bigcup_{c \in aac} G(symdiff\,(M,c), U \backslash \ulcorner c, V \backslash c \urcorner)\ \mathsf{fi}\ . \end{array} \tag{27}$$

The core of this function is the computation of the increasing alternating chains. Since an alternating chain orders the nodes in a certain way, we can solve this problem using the generalization HP of the Hamiltonian path operation H.

To this end we define

$$calc\_aac(M,U,V) \stackrel{\text{def}}{=} increasingchain(M,U,V) \triangleleft altchain(M)\ .$$

altchain computes all alternating paths of the graph corresponding to $\overleftrightarrow{R}$. The extension of R to its symmetric closure is necessary, since otherwise, by the bipartiteness assumption for R, we could not construct any alternating chains. We compute an alternating chain of length n as a Hamiltonian path of length n and specify:

$$altchain(M) \stackrel{\text{def}}{=} \bigcup_{n=2}^{|A|} HP(alter(M))(n)\ .$$

For the complete computation of *calc_aac* we have to filter out from the intermediate result those increasing chains which start in U and end in V. This is done by the assertion $increasingchain(M,U,V)(w)$:

$$\begin{array}{l} increasingchain(M,U,V)(w) \stackrel{\text{def}}{\Leftrightarrow} \\ \quad \|w\| \geq 2\ \wedge\ \ulcorner w \subseteq isolated(M) \cap U\ \wedge \\ \quad w\urcorner \subseteq isolated(M) \cap V\ . \end{array} \tag{28}$$

The computation of *altchain* with the union of all alternating chains of lengths 2 up to $|A|$ is inefficient, however, since the chains with length $n-1$ are used again for the computation of the chains with length n. Hence it is better to adapt the basic algorithm (23) so that it yields alternating chains of arbitrary length. Since a chain contains at least two nodes, we shall make the case $n = 2$ the termination case. Moreover, we change the end node set, i.e., we start the construction of the chains not with nodes from A but with nodes from $isolated(M) \cap V$, since all chains whose last nodes are not from this set will be eliminated from the result by *increasingchain* anyway.

In sum we obtain therefore:

$$
\begin{array}{l}
calc\_aac(M,U,V) = \\
\quad increasingchain(M,U,V) \triangleleft alc(M, isolated(M) \cap V)(|A|) \ , \\
\\
alc(M,S)(n) = \\
\quad \mathsf{if}\, n < 2\, \mathsf{then}\, \emptyset \\
\quad [\!]\ n = 2\, \mathsf{then}\, alter(M) \triangleleft (R \bowtie S) \\
\quad [\!]\ n > 2\, \mathsf{then\ let}\ W = alc(M,S)(n-1) \\
\quad\ \mathsf{in} \qquad W \cup \bigcup_{p \in W \wedge \|p\| = n-1} alter(M) \triangleleft (non(p) \bowtie R \bowtie p)\ \ \mathsf{fi}\ .
\end{array}
\tag{29}
$$

7 Conclusion

We have presented derivations of schematic algorithms for two classes of graph problems. In particular, the class of Hamiltonian path problems presents a wide variety of applications, among which finding maximum cardinality matchings is the most advanced. Further instances of this class can be found in [Russling 96b].

In the case of layer oriented traversals it was surprising how far the abstract framework of typed Kleene algebras carries. The axiomatization used there is much weaker than that of relational or sequential calculus.

It is to be hoped that a similar treatment of other graph algorithm classes can be found.

Acknowledgement The derivation of the maximal matching algorithm was inspired by [Berghammer]. The anoymous referees provided a number of quite helpful comments.

References

[Aho et al. 74] A.V. Aho, J.E. Hopcroft, J.D. Ullman: The design and analysis of computer algorithms. Reading, Mass.: Addison Wesley 1974

[Berghammer] R. Berghammer: unpublished manuscript

[Biggs 89] Biggs: Discrete Mathematics. Oxford: Clarendon Press 1989

[Brunn 97] T. Brunn: Deduktiver Entwurf und funktionale Programmierung von Graphenalgorithmen. Institut für Informatik, Universität Augsburg, Diplomarbeit, August 1997

[Conway 71] J.H. Conway: Regular algebra and finite machines. London: Chapman and Hall 1971

[Kleene 52] S.C. Kleene: Introduction to metamathematics. New York: van Nostrand 1952

[Möller 91] B. Möller: Relations as a program development language. In: B. Möller (ed.): Constructing programs from specifications. Proc. IFIP TC2/WG 2.1 Working Conference on Constructing Programs from Specifications, Pacific Grove, CA, USA, 13–16 May 1991. Amsterdam: North-Holland 1991, 373–397

[Möller 93] B. Möller: Derivation of graph and pointer algorithms. In: B. Möller, H.A. Partsch, S.A. Schuman (eds.): Formal program development. Proc. IFIP TC2/WG2.1 State of Art Seminar, Rio de Janeiro, Jan. 1992. Lecture Notes in Computer Science **755**. Berlin: Springer 1993, 123–160

[Möller 96] B. Möller: Assertions and recursions. In: G. Dowek, J. Heering, K. Meinke, B. Möller (eds.): Higher order algebra, logic and term rewriting. Second International Workshop, Paderborn, Sept. 21-22, 1995. Lecture Notes in Computer Science **1074**. Berlin: Springer 1996, 163–184

[Möller 98] B. Möller: Typed Kleene algebras. Institut für Informatik, Universität Augsburg, Technical Report 1998-3, April 1998

[Möller, Russling 93] B. Möller, M. Russling: Shorter paths to graph algorithms. In: R.S. Bird, C.C. Morgan, J.C.P. Woodcock (eds.): Mathematics of Program Construction. Lecture Notes in Computer Science **669**. Berlin: Springer 1993, 250–268. Extended version: Science of Computer Programming **22**, 157–180 (1994)

[Russling 96a] M. Russling: Deriving a class of layer-oriented graph algorithms. Science of Computer Programming **26**, 117-132 (1996).

[Russling 96b] M. Russling: Deriving general schemes for classes of graph algorithms. Augsburger Mathematisch-Naturwissenschaftliche Schriften, Band 13 (1996).

[Schmidt, Ströhlein 93] G. Schmidt, T. Ströhlein: Relations and graphs. Discrete Mathematics for Computer Scientists. EATCS Monographs on Theoretical Computer Science. Berlin: Springer 1993