

Layered graph traversals and Hamiltonian path problems - an algebraic approach

Thomas Brunn, Bernhard Möller, Martin Russling

Angaben zur Veröffentlichung / Publication details:

Brunn, Thomas, Bernhard Möller, and Martin Russling. 1997. "Layered graph traversals and Hamiltonian path problems - an algebraic approach." Augsburg: Institut für Informatik, Universität Augsburg.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under the following conditions:

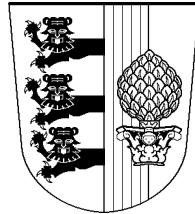
Deutsches Urheberrecht

Weitere Informationen finden Sie unter: / For more information see:

<https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publizieren>



UNIVERSITÄT AUGSBURG

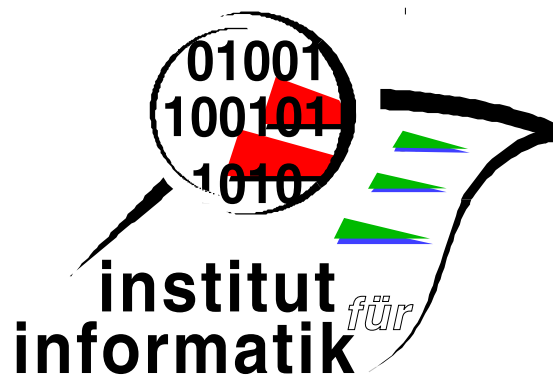


**Layered Graph Traversals and
Hamiltonian Path Problems –
An Algebraic Approach**

Thomas Brunn Bernhard Möller
Martin Russling

Report 1997-08

Dezember 1997



INSTITUT FÜR INFORMATIK

D-86135 AUGSBURG

Copyright © Thomas Brunn
Bernhard Möller
Martin Russling
Institut für Informatik
Universität Augsburg
D-86135 Augsburg, Germany
<http://www.Informatik.Uni-Augsburg.DE>
— all rights reserved —

Layered Graph Traversals and Hamiltonian Path Problems – An Algebraic Approach

Thomas Brunn¹, Bernhard Möller², and Martin Russling³

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, cycle detection 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.

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

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 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 $A^+ \stackrel{\text{def}}{=} A^* \setminus \varepsilon$. 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 $\|w\|$, while the set of letters occurring in w is denoted by $\text{set}(w)$.

A *relation* is a language R in which all words have equal length. This length is called the *arity* of the relation, in symbols $\text{ar } R$. 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

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

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 $f : A^* \rightarrow \mathcal{P}(A^*)$ is denoted by the same symbol f and has the functionality $f : \mathcal{P}(A^*) \rightarrow \mathcal{P}(A^*)$. 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.

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

$$\varepsilon \bowtie s = \emptyset = s \bowtie \varepsilon, \quad \varepsilon ; s = \emptyset = s ; \varepsilon, \tag{1}$$

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

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

These operations provide two different ways of “glueing” 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 $R \subseteq A \bullet A$ modelling 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 glueing 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

$$R, R \bowtie R, R \bowtie (R \bowtie R), \dots$$

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

$$R, R ; R, R ; (R ; R), \dots$$

just state existence of these paths between pairs of vertices.

An interesting example of the pointwise extension of composition is the test whether the intersection of two unary relations S and T is empty:

$$S ; T = \begin{cases} \varepsilon & \text{if } S \cap T \neq \emptyset , \\ \emptyset & \text{if } S \cap T = \emptyset , \end{cases} \quad (2)$$

if $\text{ar } T = \text{ar } S = 1$.

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

$$S \bowtie T = S \cap T . \quad (3)$$

3 Kleene Algebras and Closures

3.1 Kleene Algebras

A Kleene algebra is a quintuple $(S, \Sigma, \cdot, 0, 1)$ consisting of a set S , operations $\Sigma : \mathcal{P}(S) \rightarrow S$ and $\cdot : S \bullet S \rightarrow S$ 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 :

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

Examples of Kleene algebras are given by

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

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

$$x \leq y \stackrel{\text{def}}{\Leftrightarrow} x + y = y , \tag{4}$$

where

$$x + y \stackrel{\text{def}}{=} \Sigma\{x, y\}. \tag{5}$$

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 \cdot^* by

$$x^* \stackrel{\text{def}}{=} \mu y . 1 + x \cdot y , \tag{6}$$

and a proper closure operator \cdot^+ by

$$x^+ \stackrel{\text{def}}{=} \mu y . x + x \cdot y . \tag{7}$$

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

$$R^{\rightsquigarrow} = \bigcup_{i \in \mathbb{N}} {}^i R, \quad (8)$$

where

$${}^0 R \stackrel{\text{def}}{=} \varepsilon, \quad (9)$$

$${}^1 R \stackrel{\text{def}}{=} A, \quad (10)$$

$${}^{i+1} R \stackrel{\text{def}}{=} R \bowtie {}^i R \quad (i > 0). \quad (11)$$

Hence ${}^i R$ 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 call a pretyped 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 families $(x_j)_{j \in J}$ of subidentities and all $a \in S$ that

$$\left(\bigcap_{j \in J} x_j\right) \cdot a = \bigcap_{j \in J} x_j \cdot a \quad \wedge \quad a \cdot \left(\bigcap_{j \in J} x_j\right) = \bigcap_{j \in J} a \cdot x_j.$$

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

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

By this, the operations $\langle _$ and $_ \rangle$ are universally disjunctive and hence monotonic and strict. Moreover, we can show the usual properties of domain and co-domain (see [Möller 97]):

$$\begin{aligned} \langle a &= \bigcap \{x : x \leq 1 \wedge x \cdot a = a\}, \\ a \rangle &= \bigcap \{y : x \leq 1 \wedge a \cdot y = a\}, \\ \langle 1 &= 1, \quad 1 \rangle = 1, \\ \langle \langle a &= \langle a, \quad a \rangle \rangle = a \rangle, \\ \langle (a \cdot b) &= \langle (b \cdot \langle a, \quad (a \cdot b) \rangle) = \langle a \rangle \cdot b \rangle, \\ \langle a \leq \langle b &\Rightarrow \langle (a \cdot c) \leq \langle (b \cdot c), \quad a \rangle \leq b \rangle \Rightarrow (a \cdot c) \rangle \leq (b \cdot c) \rangle. \end{aligned} \quad (12)$$

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

$$\langle R = R; \top \cap I \wedge R \rangle = \top; R \cap I.$$

In *LAN* we have

$$\langle U = U \rangle = \begin{cases} \varepsilon & \text{if } U \neq \emptyset , \\ \emptyset & \text{otherwise.} \end{cases}$$

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

$$\langle U = \text{first}(U) \wedge U \rangle = \text{last}(U) ,$$

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

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

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”, as was already apparent in (2). Expressions that yield one of these values are therefore also called *assertions*. The assertion 0 means not only “false”, but also “undefined”. Negation is defined by

$$\neg 0 \stackrel{\text{def}}{=} 1 , \quad \neg 1 \stackrel{\text{def}}{=} 0 . \quad (13)$$

Then for an assertion b and an element c we have

$$b \cdot c = c \cdot b = \begin{cases} c & \text{if } b = 1 , \\ 0 & \text{if } b = 0 . \end{cases} \quad (14)$$

The conjunction of 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:

$$\text{if } b \text{ then } c \text{ else } d \text{ fi} \stackrel{\text{def}}{=} b \cdot c \cup \neg b \cdot d \quad (15)$$

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

$$\text{if } b \text{ then } d \text{ else if } c \text{ then } d \text{ else } e \text{ fi} = \text{if } b \vee c \text{ then } d \text{ else } e \text{ fi} \quad (16)$$

for assertions b, c and elements d, e .

3.5 Filters

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

$$B \triangleleft w = B(w) \bullet w = \begin{cases} w & \text{if } B(w) = \varepsilon , \\ \emptyset & \text{if } B(w) = \emptyset . \end{cases} \quad (17)$$

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

$$B \triangleleft W = \bigcup_{w \in W} B(w) \bullet w . \quad (18)$$

This operation distributes through \cap and \cup :

$$B \triangleleft (V \cap W) = B \triangleleft V \cap B \triangleleft W , \quad (19)$$

$$B \triangleleft (V \cup W) = B \triangleleft V \cup B \triangleleft W . \quad (20)$$

4 A General Graph Problem Class

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

$$E(W)(f, g) \stackrel{\text{def}}{=} g(f(W)) \quad (21)$$

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.
- $f : A^* \rightarrow M$ 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.
- $g : \mathcal{P}(M) \rightarrow \mathcal{P}(M)$ 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

$$\begin{aligned} (\text{GEN1}) \quad & g(K) \subseteq K , \\ (\text{GEN2}) \quad & g(K \cup L) = g(g(k) \cup g(L)) \text{ (weak distributivity),} \end{aligned}$$

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

$$(\text{GEN2}') \quad g(K \cup L) = g(K \cup g(L)).$$

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

$$E(W)(f, g) \stackrel{\text{def}}{=} g(f(w)) \quad (22)$$

where

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

$$\begin{aligned} (\text{GEN1}) \quad & g(K) \subseteq K, \\ (\text{GEN2}) \quad & g(K \cup L) = g(g(K) \cup g(L)) \text{ (weak distributivity),} \end{aligned}$$

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 (22) $W = S \bowtie R^{\rightsquigarrow} \bowtie T$ and specify a graph traversal operation F by

$$F(f, g)(S, R, T) \stackrel{\text{def}}{=} E(f, g)(S \bowtie R^{\rightsquigarrow} \bowtie T) = g(f(S \bowtie R^{\rightsquigarrow} \bowtie T)), \quad (23)$$

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

$$F(f, g)(a, b, c) \stackrel{\text{def}}{=} E(f, g)(a \cdot b^* \cdot c) = g(f(a \cdot b^* \cdot c)), \quad (24)$$

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

$$\begin{aligned}
& F(f, g)(a, b, c) \\
= & \quad \{ \text{definition} \} \\
& g(f(a \cdot b^* \cdot c)) \\
= & \quad \{ \text{idempotence of } + \} \\
& g(f(a \cdot b^* \cdot c) + f(a \cdot b^* \cdot c)) \\
= & \quad \{ (6) \} \\
& g(f(a \cdot (1 + b \cdot b^*) \cdot c) + f(a \cdot b^* \cdot c)) \\
= & \quad \{ \text{distributivity and neutrality} \} \\
& g(f(a \cdot c + a \cdot b \cdot b^* \cdot c) + f(a \cdot b^* \cdot c)) \\
= & \quad \{ \text{associativity of } + \text{ and disjunctivity of } f \text{ twice} \} \\
& g(f(a \cdot c) + f(a \cdot b \cdot b^* \cdot c + a \cdot b^* \cdot c)) \\
= & \quad \{ \text{distributivity} \} \\
& g(f(a \cdot c) + f((a \cdot b + a) \cdot b^* \cdot c)) \\
= & \quad \{ (\text{GEN2}') \} \\
& g(f(a \cdot c) + g(f((a + a \cdot b) \cdot b^* \cdot c))) \\
= & \quad \{ \text{definition} \} \\
& g(f(a \cdot c) + F(f, g)(a + a \cdot b, b, c)) .
\end{aligned}$$

5.3 Termination Cases

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

$$\begin{aligned}
(\text{LAY1}) \quad \langle c \leq a \rangle & \Rightarrow g(f(a \cdot c)) = g(f(a \cdot c) + f(a \cdot u \cdot c)) , \\
(\text{LAY2}) \quad \langle a \cdot u \rangle \leq a & \Rightarrow g(f(a \cdot c)) = g(f(a \cdot c) + f(a \cdot u \cdot c)) ,
\end{aligned}$$

with $a, c, u \in S$. These two conditions are used to obtain two termination cases of the algorithm.

When dealing with graph problems, $\langle c \leq a \rangle$ is the case where the set of starting nodes of c already is contained in the set of end nodes of a . $\langle a \cdot c \rangle \leq a$ means that by further traversal of the graph no new end nodes are reached any more. In both cases the second use of the abstraction f should give no new information and be ignored by g .

Case 1: $\langle c \leq a \rangle$. Then we get

$$\begin{aligned}
& F(f, g)(a, b, c) \\
= & \quad \{ \text{definition} \} \\
& g(f(a \cdot b^* \cdot c)) \\
= & \quad \{ \text{by (6) we have } b^* = 1 + b^* \} \\
& g(f(a \cdot (1 + b^*) \cdot c)) \\
= & \quad \{ \text{distributivity and neutrality} \} \\
& g(f(a \cdot c + a \cdot b^* \cdot c)) \\
= & \quad \{ \text{disjunctivity of } f \} \\
& g(f(a \cdot c) + f(a \cdot b^* \cdot c)) \\
= & \quad \{ \text{(LAY1)} \} \\
& g(f(a \cdot c)).
\end{aligned}$$

For the second case we need the following lemma:

Lemma 1. For $a, b \in S$ one has $(a \cdot b) \leq a \Rightarrow (a \cdot b^*) \leq a$.

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

$$\begin{aligned}
& (a \cdot (b \cdot c)) \\
= & \quad \{ \text{associativity} \} \\
& ((a \cdot b) \cdot c) \\
\leq & \quad \{ \text{assumption and (12)} \} \\
& (a \cdot c) \\
\leq & \quad \{ P[c] \} \\
& a
\end{aligned}$$

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

Hence we obtain

Case 2: $(a \cdot b) \leq a$.

$$\begin{aligned}
& F(f, g)(a, c) \\
= & \quad \{ \text{definition} \} \\
& g(f(a \cdot b^* \cdot c)) \\
= & \quad \{ \text{by (6) one has } b^* = 1 + b^* \} \\
& g(f(a \cdot (1 + b^*) \cdot c)) \\
= & \quad \{ \text{distributivity and neutrality} \}
\end{aligned}$$

$$\begin{aligned}
& g(f(a \cdot c) + f(a \cdot b^* \cdot c)) \\
= & \{ \text{Lemma 1, (LAY2)} \} \\
& g(f(a \cdot c)) .
\end{aligned}$$

In sum we have derived the following basic algorithm:

$$\begin{aligned}
F(f, g)(a, b, c) = & \text{if } \langle c \leq a \rangle \vee \langle a \cdot b \rangle \leq a \\
& \text{then } g(f(a \cdot c)) \\
& \text{else } g(f(a \cdot c) + F(f, g)(a + a \cdot b, b, c)) \text{ fi} .
\end{aligned} \tag{25}$$

5.4 Applications

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

$$reach(S) \stackrel{\text{def}}{=} F(_, id)(S, R, A) . \tag{26}$$

It is easily checked that f and g satisfy the required conditions.

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

$$\begin{aligned}
& reach(S) \\
= & \{ \text{definition} \} \\
& F(_, id)(S, R, A) \\
= & \{ (25) \} \\
& \text{if } \langle A \subseteq S \rangle \vee \langle S \bowtie R \rangle \subseteq S \\
& \quad \text{then } (S \bowtie A) \\
& \quad \text{else } (S \bowtie A) \cup reach(S \cup S \bowtie R) \text{ fi} \\
= & \{ (3), (12) \} \\
& \text{if } A \subseteq S \vee \langle S \bowtie R \rangle \subseteq S \\
& \quad \text{then } S \\
& \quad \text{else } S \cup reach(S \cup S \bowtie R) \text{ fi} \\
= & \{ A \subseteq S \Rightarrow A = S \Rightarrow \langle S \bowtie R \rangle \subseteq S \} \\
& \text{if } \langle S \bowtie R \rangle \subseteq S \\
& \quad \text{then } S \\
& \quad \text{else } S \cup reach(S \cup S \bowtie R) \text{ fi} .
\end{aligned}$$

Shortest Connecting Path We define

$$shortestpaths(a, c) \stackrel{\text{def}}{=} F(id, minpaths)(a, c) , \tag{27}$$

with

$$\begin{aligned}
\text{minpaths}(a) &\stackrel{\text{def}}{=} \text{let } ml = \min(\bigcup_{x \in a} \|x\|) \\
&\text{in } \bigcup_{x \in a} \text{if } \|x\| = ml \text{ then } x \text{ else } \emptyset .
\end{aligned} \tag{28}$$

Here *pathminlength* selects from a set of words the ones with the least number of letters. Again the conditions (GEN) and (LAY) are satisfied. Therefore we have the following algorithm for computing the shortest path between a set S and the node y :

```

shortestpaths( $S, y$ )
=   { definition }
     $F(id, \text{minpaths})(S, y)$ 
=   { (25) }
    if  $\langle y \subseteq S \rangle \vee \langle S \bowtie R \rangle \subseteq S$ 
      then  $\text{minpaths}(S \bowtie y)$ 
      else  $\text{minpaths}(S \bowtie y \cup \text{shortestpaths}(S \cup S \bowtie R, y))$  fi
=   { (16) }
    if  $y \in S$ 
      then  $\text{minpaths}(S \bowtie y)$ 
      else if  $\langle S \bowtie R \rangle \subseteq S$ 
          then  $\text{minpaths}(S \bowtie y)$ 
          else  $\text{minpaths}(S \bowtie y \cup \text{shortestpaths}(S \cup S \bowtie R, y))$  fi fi
=   {  $y \notin S \Rightarrow S \bowtie y = \emptyset$  }
    if  $y \in S$ 
      then  $\text{minpaths}(S \bowtie y)$ 
      else if  $\langle S \bowtie R \rangle \subseteq S$ 
          then  $\text{minpaths}(\emptyset)$ 
          else  $\text{minpaths}(\text{shortestpaths}(S \cup S \bowtie R, y))$  fi fi
=   { definition and idempotence of  $\text{minpaths}$  }
    if  $y \in S$ 
      then  $\text{minpaths}(S \bowtie y)$ 
      else if  $\langle S \bowtie R \rangle \subseteq S$ 
          then  $\emptyset$ 
          else  $\text{shortestpaths}(S \cup S \bowtie R, y)$  fi fi .

```

5.5 Improved Efficiency

In graph applications, the parameter a in algorithm (25) 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

$$F_{\text{eff}}(f, g)(a, b, c, u) = (a \sqcap u) = \emptyset \cdot F(f, g)(a + u, b, c). \quad (29)$$

From this we get immediately

$$F(f, g)(a, b, c) = F_{\text{eff}}(f, g)(a, b, c, \emptyset). \quad (30)$$

Let now $v \stackrel{\text{def}}{=} a + u$. By (25) and (30) we get the following special case:

$$\begin{aligned} \langle c \leq v \rangle \vee ((a + u) \cdot b) \leq v &\Rightarrow \\ F_{\text{eff}}(f, g)(a, c, u) &= g(f(v \cdot c)). \end{aligned} \quad (31)$$

The recursive case looks as follows:

$$\begin{aligned} &F_{\text{eff}}(f, g)(a, b, c, u) \\ &= \quad \{ \text{definitions} \} \\ &\quad F(f, g)(v, b, c) \\ &= \quad \{ (25) \} \\ &\quad g(f(v \cdot c) + g(f((v + v \cdot b) \cdot b^* \cdot c))) \\ &= \quad \{ \text{let } b_1 = b \cdot v, b_2 = b \setminus b_1 \} \\ &\quad g(f(v \cdot b) + g(f((v + v \cdot (b_1 + b_2)) \cdot b^* \cdot c))) \\ &= \quad \{ \text{distributivity and disjunctivity of } f, \text{ several times (GEN2')} \} \\ &\quad g(f(v \cdot b) + f(v \cdot b_2 \cdot b^* \cdot c) + g(f(v \cdot b^* \cdot c) + f(v \cdot b_1 \cdot b^* \cdot c))) \\ &= \quad \{ \text{definition of } b_1, (\text{LAY2}) \} \\ &\quad g(f(v \cdot b) + f(v \cdot b_2 \cdot b^* \cdot c) + g(f(v \cdot b^* \cdot c))) \\ &= \quad \{ (\text{GEN2}') \text{ and distributivity} \} \\ &\quad g(f(v \cdot b) + g(f((v + v \cdot b_2) \cdot b^* \cdot c))) \\ &= \quad \{ \text{definition} \} \\ &\quad g(f(v \cdot b) + F(f, g)(v + v \cdot b_2, b, c)) \\ &= \quad \{ \text{definition} \} \\ &\quad g(f(v \cdot b) + F_{\text{eff}}(f, g)(v \cdot b_2, b, c, v)). \end{aligned}$$

In sum we get

$$\begin{aligned} F_{\text{eff}}(f, g)(a, b, c, u) &= \\ \text{let } v &= a + u \\ b_1 &= b \cdot v \end{aligned} \quad (32)$$

```

    b2 = b \ b1
  in (a) □ u = ∅ .
    if ⟨c ≤ v⟩ ∨ ⟨v · b⟩ ≤ v
      then g(f(v · c))
    else g(f(v · c) + Feff(f, g)(v · b2, b, c, v)) fi .

```

It is easily checked 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

$$\begin{aligned}
 F_{\text{eff}^2}(f, g)(a, b, c, u) = & \\
 \text{let } v = a + u & \\
 b_1 = b \cap A \cdot v & \\
 b_2 = b \setminus b_1 & \\
 \text{in } (a) \square u = \emptyset \wedge (u \cdot b_2 \leq a + u) \cdot & \\
 \text{if } \langle c \leq v \rangle \vee \langle a \cdot b_2 \rangle \leq v & \\
 \text{then } g(f(v \cdot c)) & \\
 \text{else } g(f(v \cdot c) + F_{\text{eff}^2}(f, g)(a \cdot b_2, b, c, v)) \text{ fi} . &
 \end{aligned} \tag{33}$$

Hence 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 permutation of the node set or by a binary relation on the set. Since this can be computed by subsets of the set of Hamiltonian paths, the latter will serve as the basis of the further computations.

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 node of the respective graph exactly once. Hence the set of Hamiltonian paths is defined as follows:

$$\text{hamiltonianpaths} \stackrel{\text{def}}{=} \text{perms}(A) \cap R^{\rightsquigarrow}, \tag{34}$$

with

$$\text{perms}(S) \stackrel{\text{def}}{=} \bigcup_{x \in S} x \bullet \text{perms}(S \setminus x). \tag{35}$$

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 assertion B only to the final paths. 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

$$B(v \bullet w) \subseteq B(w) , \quad (36)$$

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.

We choose therefore $W \stackrel{\text{def}}{=} \text{perms}(A) \cap R^{\rightsquigarrow}$, $f \stackrel{\text{def}}{=} id$ and $g \stackrel{\text{def}}{=} B \triangleleft$ and obtain in this way from (22) the Hamiltonian path operation H by

$$H(B) \stackrel{\text{def}}{=} E(\text{perms}(A) \cap R^{\rightsquigarrow})(id, B) = B \triangleleft \text{hamiltonianpaths} , \quad (37)$$

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:

$$\text{hamiltonianpaths} = \text{hp}(|A|). \quad (38)$$

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

$$\text{hp}(n) \stackrel{\text{def}}{=} \text{partperms}(n, A) \cap R^{\rightsquigarrow} , \quad (39)$$

where $\text{partperms}(n, S)$ is the set of partial permutations of length n of a set S :

$$\text{partperms}(n, S) \stackrel{\text{def}}{=} \bigcup_{T \subseteq S \cap |T|=n} \text{perms}(T). \quad (40)$$

One sees easily that $n > |S|$ implies $\text{partperms}(n, S) = \emptyset$ and hence

$$\text{hp}(n) = \emptyset . \quad (41)$$

Moreover, $n \leq |S|$ implies $\text{partperms}(n, S) = n$ and hence

$$\text{hp}(n) = \text{partperms}(n, A) \cap {}^n R . \quad (42)$$

Finally we obtain

$$\bigcup_{i \in \mathbb{N}} \text{hp}(i) = \bigcup_{0 \leq i \leq |A|} \text{hp}(i) . \quad (43)$$

For the transformation into recursive form we need two auxiliary lemmas:

Lemma 2. For all $S \subseteq A$ and all $k \leq n \leq |S|$ one has

$$partperms(n, S) = \bigcup_{T \subseteq S \wedge |T|=k} \bigcup_{U \subseteq S \setminus T \wedge |U|=n-k} perms(U) \bullet perms(T).$$

Proof: siehe [Russling 96b]. ■

Lemma 3. For $n \neq 0$ one has

$$partperms(n, S) = \bigcup_{T \subseteq S \wedge |T|=n-1} (S \setminus T) \bullet perms(T).$$

Proof: Set $k = n - 1$ in Lemma 2. ■

Now we can perform a case distinction for *partperms*:

Case 1: $n = 0$.

$$\begin{aligned} & partperms(n, S) \\ = & \{ \text{definition} \} \\ & perms(\emptyset) \\ = & \{ \text{definition} \} \\ & \varepsilon \end{aligned}$$

Case 2: $n \geq 1$.

$$\begin{aligned} & partperms(n, S) \\ = & \{ \text{Lemma 3} \} \\ & \bigcup_{T \subseteq S \wedge |T|=n-1} (S \setminus T) \bullet perms(T) \\ = & \{ \text{pointwise extension and } p \in perms(T) \Rightarrow set(p) = T \} \\ & \bigcup_{T \subseteq S \wedge |T|=n-1} \bigcup_{p \in perms(T)} (S \setminus set(p)) \bullet p \\ = & \{ \text{boolean algebra} \} \\ & \bigcup_{p \in \bigcup_{T \subseteq S \wedge |T|=n-1} perms(T)} (S \setminus set(p)) \bullet p \\ = & \{ \text{definition} \} \\ & \bigcup_{p \in partperms(n-1, S)} (S \setminus set(p)) \bullet p \end{aligned}$$

We extend the case distinction for *hp* by the case $n = 1$:

Case 1: $n = 0$.

$$\begin{aligned}
& hp(n) \\
= & \{ (42) \} \\
& partperms(0, A) \cap {}^0R \\
= & \{ \text{case distinction of } partperms \} \\
& \varepsilon.
\end{aligned}$$

Case 2: $n = 1$.

$$\begin{aligned}
& hp(n) \\
= & \{ (42) \} \\
& partperms(1, A) \cap {}^1R \\
= & \{ \text{case distinction of } partperms \text{ and (10)} \} \\
& \varepsilon.
\end{aligned}$$

For the next case we need

Lemma 4. 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: $n > 1$.

$$\begin{aligned}
& hp(n) \\
= & \{ (42) \} \\
& partperms(n, A) \cap {}^nR \\
= & \{ \text{case distinction of } partperms \text{ and (11)} \} \\
& \bigcup_{p \in partperms(n-1, A)} (A \setminus set(p)) \bullet p \cap R \bowtie {}^{n-1}R \\
= & \{ \text{distributivity} \} \\
& \bigcup_{p \in partperms(n-1, A)} (A \setminus set(p)) \bullet p \cap R \bowtie {}^{n-1}R \\
= & \{ \text{Lemma 4} \} \\
& \bigcup_{p \in partperms(n-1, A)} (A \setminus set(p)) \bullet p \bowtie R \bowtie (p \cap {}^{n-1}R) \\
= & \{ \text{by case distinction } p \in {}^{n-1}R \} \\
& \bigcup_{p \in partperms(n-1, A) \cap {}^{n-1}R} (A \setminus set(p)) \bowtie R \bowtie p \\
= & \{ (42) \}
\end{aligned}$$

$$\bigcup_{p \in hp(n-1)} (A \setminus set(p)) \bowtie R \bowtie p.$$

In sum we get for hp :

$$hp(n) = \begin{cases} \text{if } n = 0 \text{ then } \varepsilon \\ \text{if } n = 1 \text{ then } A \\ \text{if } n > 1 \text{ then } \bigcup_{p \in hp(n-1)} (A \setminus set(p)) \bowtie R \bowtie p \text{ fi.} \end{cases} \quad (44)$$

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 :

$$HP(B)(n) \stackrel{\text{def}}{=} B \triangleleft hp(n). \quad (45)$$

Because of

$$H(B) = HP(B)(|A|). \quad (46)$$

we perform also for HP a case distinction:

Case 1: $n = 0$.

$$\begin{aligned} & HP(B)(n) \\ &= \{ \text{definition} \} \\ & B \triangleleft hp(0) \\ &= \{ (44) \} \\ & B \triangleleft \varepsilon. \end{aligned}$$

Case 2: $n = 1$.

$$\begin{aligned} & HP(B)(n) \\ &= \{ \text{definition} \} \\ & B \triangleleft hp(1) \\ &= \{ (44) \} \\ & B \triangleleft A. \end{aligned}$$

Case 3: $n > 1$.

$$\begin{aligned} & HP(B)(n) \\ &= \{ \text{definition} \} \\ & B \triangleleft hp(n) \\ &= \{ (44) \} \end{aligned}$$

$$\begin{aligned}
& B \triangleleft \bigcup_{p \in hp(n-1)} (A \setminus set(p)) \bowtie R \bowtie p \\
= & \quad \{ \text{distributivity} \} \\
& \bigcup_{p \in hp(n-1)} B \triangleleft ((A \setminus set(p)) \bowtie R \bowtie p) \\
= & \quad \{ \text{(HAM) } B \text{ is suffix closed} \Rightarrow \\
& \quad \text{for } q \in (A \setminus set(p)) \bowtie R \bowtie p : B(q) \subseteq B(p) \} \\
& \bigcup_{p \in hp(n-1)} (B \triangleleft ((A \setminus set(p)) \bowtie R \bowtie p)) \bullet B(p) \\
= & \quad \{ \text{by case distinction } B(p) = \varepsilon \} \\
& \bigcup_{p \in B \triangleleft hp(n-1)} B \triangleleft ((A \setminus set(p)) \bowtie R \bowtie p) \\
= & \quad \{ \text{definition} \} \\
& \bigcup_{p \in HP(n-1)} B \triangleleft ((A \setminus set(p)) \bowtie R \bowtie p).
\end{aligned}$$

In sum we have

$$\begin{aligned}
HP(B)(n) = & \text{if } n = 0 \text{ then } B \triangleleft \varepsilon \\
& \square n = 1 \text{ then } B \triangleleft A \\
& \square n > 1 \text{ then } \bigcup_{p \in HP(B)(n-1)} B \triangleleft ((A \setminus set(p)) \bowtie R \bowtie p) \text{ fi.}
\end{aligned} \tag{47}$$

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 B is maintained. 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.

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 tackle the problem from behind 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:

$$allowedrel(\varepsilon) \stackrel{\text{def}}{=} \emptyset, \quad (48)$$

$$allowedrel(a \bullet w) \stackrel{\text{def}}{=} a \bullet set(w) \cup allowedrel(w), \quad (49)$$

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

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

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

$$consistent(Q)(w) \stackrel{\text{def}}{=} (Q \subseteq allowedrel(w) \cup (A \setminus (set(w)) \bullet A)). \quad (50)$$

Note that for $w \in perms(A)$ one obviously has

$$consistent(Q)(w) = (Q \subseteq allowedrel(w)). \quad (51)$$

Finally we define 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

$$topsort \stackrel{\text{def}}{=} H(consistent(Q)) = consistent(Q) \triangleleft perms(A). \quad (52)$$

Using (46) we define

$$conshp(n) \stackrel{\text{def}}{=} HP(consistent(Q))(n) = consistent(Q) \triangleleft hp(n). \quad (53)$$

Hence have we the following result:

$$topsort = conshp(|A|),$$

$$\begin{aligned} conshp(n) = & \text{if } n = 0 \text{ then } B \triangleleft \varepsilon \\ & \square n = 1 \text{ then } consistent(Q) \triangleleft A \\ & \square n > 1 \text{ then } \bigcup_{p \in conshp(n-1)} consistent(Q) \triangleleft ((A \setminus set(p)) \bowtie R \bowtie p) \text{ fi.} \end{aligned} \quad (54)$$

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 $R \subseteq U \bullet V$.

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, $M \subseteq R$ and $\bigcup_{m \in M} ends(M \setminus m) \cap ends(m) = \emptyset$, where

$$ends(\varepsilon) = \emptyset \quad (55)$$

$$ends(a) = \{a\} \quad (56)$$

$$ends(a \bullet u \bullet b) = \{a\} \cup \{b\}, \quad (57)$$

for $a, b \in A, u \in A^*$. The *maximal matching problem* consists in finding matchings with maximal cardinality $|M|$.

By $\overset{\leftrightarrow}{M}$ we now denote the symmetric closure of M . A node is *isolated* if it is not touched by any edge in M . A chain $\{x_0, x_1, \dots, x_n\}$ of edges is *alternating* w.r.t. M if the edges $\{x_i, x_{i+1}\}$ alternately lie within and without of $\overset{\leftrightarrow}{M}$. An alternating chain is *increasing* if its extremal nodes x_0 and x_n are isolated. Note that the length of an increasing and alternating 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 alternating 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 *symdiff*:

$$\begin{aligned} \text{symdiff}(M, c) &\stackrel{\text{def}}{=} \\ &\text{let } x = \langle c \\ &\quad y = \langle \text{rest}(c) \\ &\text{in if } \|c\| \leq 2 \\ &\quad \text{then } M \cup \{x, y\} \\ &\quad \text{else let } z = \langle \text{rest}(\text{rest}(c)) \\ &\quad \quad \text{in } \text{symdiff}(M \setminus \{z \bullet y\} \cup \{x \bullet y\}, \text{rest}(\text{rest}(c))) \text{ fi.} \end{aligned} \quad (58)$$

The fact that using an increasing, alternating 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 alternating 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 5. A matching has maximal cardinality iff no increasing alternating chain exists.

For the proof see e.g. [Biggs 89]. We have therefore the specification

$$\text{maxmatch} \stackrel{\text{def}}{=} G(\emptyset, U, V) \quad (59)$$

$$\begin{aligned}
& G(M, U, V) \stackrel{\text{def}}{=} \\
& \text{let } aac = \text{calc_aac}(M, U, V) \\
& \text{in if } aac = \emptyset \\
& \quad \text{then } M \\
& \quad \text{else } \bigcup_{c \in aac} G(\text{symdiff}(M, c), U \setminus \langle c, V \setminus c \rangle) .
\end{aligned} \tag{60}$$

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

$$\text{calc_aac}(M, U, V) \stackrel{\text{def}}{=} \text{augmentingchain}(M, U, V) \triangleleft \text{altchain}(M). \tag{61}$$

altchain computes all alternating paths of the graph corresponding to $\overset{\leftrightarrow}{R}$. 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:

$$\text{altchain}(M) \stackrel{\text{def}}{=} \bigcup_{n=2}^{|A|} HP(\text{alter}(M))(n). \tag{62}$$

The assertion $\text{alter}(M)(w)$ is inductively defined by

$$\begin{aligned}
& \text{alter}(M)(\varepsilon) \stackrel{\text{def}}{\Leftrightarrow} \emptyset, \\
& \text{alter}(M)(a) \stackrel{\text{def}}{\Leftrightarrow} \varepsilon, \\
& \text{alter}(M)(a \bullet b) \stackrel{\text{def}}{\Leftrightarrow} a \bullet b \in \overset{\leftrightarrow}{R} \\
& \text{alter}(M)(a \bullet b \bullet c) \stackrel{\text{def}}{\Leftrightarrow} a \bullet b \in \overset{\leftrightarrow}{R} \setminus \overset{\leftrightarrow}{M} \wedge b \bullet c \in \overset{\leftrightarrow}{M} \vee \\
& \quad (a \bullet b \in \overset{\leftrightarrow}{M} \wedge b \bullet c \in \overset{\leftrightarrow}{R} \setminus \overset{\leftrightarrow}{M}) \\
& \text{alter}(M)(a \bullet b \bullet c \bullet u) \stackrel{\text{def}}{\Leftrightarrow} \text{alter}(M)(a \bullet b \bullet c) \wedge \\
& \quad \text{alter}(M)(b \bullet c \bullet u),
\end{aligned} \tag{63}$$

for $a, b, c \in A$ and $u \in A^*$.

By this inductive definition, $\text{alter}(M)$ obviously satisfies (36) and hence the condition (HAM) that $\text{alter}(M)$ must be suffix closed.

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 $\text{augmentingfilter}(M, U, V)(w)$:

$$\begin{aligned}
& \text{augmentingchain}(M, U, V)(w) \stackrel{\text{def}}{\Leftrightarrow} \\
& \|w\| \geq 2 \wedge \langle w \in (\text{exposed}(M) \cap U) \wedge \\
& w \rangle \in (\text{exposed}(M) \cap V) .
\end{aligned} \tag{64}$$

Here *exposed* gives the nodes which are isolated w.r.t. M :

$$exposed(M) \stackrel{\text{def}}{=} A \setminus ends(M). \quad (65)$$

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 (47) 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 $exposed(M) \cap V$, since all chains whose last nodes are not from this set will be eliminated from the result by *augmenting filter* anyway.

In sum we obtain therefore:

$$\begin{aligned} calc_aac(M, U, V) &= \\ & \quad augmentingchain(M, U, V) \triangleleft altchain(M, exposed(M) \cap V) \\ \\ altchain(M, S)(n) &= \\ & \quad \text{if } n < 2 \text{ then } \emptyset \\ & \quad \square n = 2 \text{ then } alter(M) \triangleleft (R \bowtie S) \\ & \quad \square n > 2 \text{ then let } W = altchain(M, S)(n - 1) \\ & \quad \quad \text{in } W \cup \bigcup_{p \in W \wedge \|p\| = n - 1} alter(M) \triangleleft ((A \setminus set(p)) \bowtie R \bowtie p). \end{aligned} \quad (66)$$

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 suprising 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].

References

- [Berghammer] R. Berghammer: unpublished manuscript
- [Biggs 89] Biggs: Discrete Mathematics. Oxford: Clarendon Press 1989

- [Brunn 97] T. Brunn: Deduktiver Entwurf und funktionale Programmierung von Graphenalgorithmien. Institut für Informatik, Universität Augsburg, Diplomarbeit, August 1997
- [Jungnickel 94] D. Jungnickel: Graphen, Netzwerke und Algorithmen. 3rd edition. Mannheim: BI Wissenschaftsverlag 1994.
- [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 93a] B. Möller: Algebraic calculation of graph and sorting algorithms. Invited Paper at the International Conference on Formal methods in Programming and their Applications, Novosibirsk, 28.6.–3.7. 1993. Lecture Notes in Computer Science **735**. Berlin: Springer 1993
- [Möller 93b] 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. Berlin: Springer
- [Möller 95] B.Möller (ed.): Mathematics of program construction. Proc. 3rd International Conference on the mathematics of program construction, Kloster Irsee, Germany, 17-21 July, 1995. Lecture Notes in Computer Science **947**. Berlin: Springer.
- [Möller 97] B. Möller: Typed Kleene algebras. Institut für Informatik, Universität Augsburg, Technical Report (forthcoming)
- [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)
- [Ottmann, Widmayer 93] T. Ottmann, P. Widmayer: Algorithmen und Datenstrukturen. Zürich : BI Wissenschaftsverlag 1993.
- [Russling 94] M. Russling: An algebraic treatment of graph and sorting algorithms. Proc. 14th Int. SCCC Conference, Concepcion, Chile, 31 October - 4 November 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