ELSEVIER

Science of Computer Programming 22 (1994) 157–180

Science of Computer Programming

Shorter paths to graph algorithms

Bernhard Möller*, Martin Russling

Institut für Mathematik, Universität Augsburg, Universitätsstr. 8, D-86135 Augsburg, Germany

Communicated by C. Morgan; revised October 1993

Abstract

We illustrate the use of formal languages and relations in compact formal derivations of some graph algorithms.

1. Introduction

The transformational or calculational approach to program development has by now a long tradition (see [1,2,4,5,12]). In it, one starts from a (possibly nonexecutable) specification and transforms it into a (hopefully efficient) program using semantics-preserving rules. Many derivations, however, suffer from the use of lengthy expressions involving formulae from predicate calculus. However, in particular in the case of graph algorithms the calculus of formal languages and relations allows considerable compactification. We use a simplified and straightened version of the framework introduced in [14] to illustrate this with derivations of algorithms for computing the length of a shortest path between two graph vertices and for cycle detection.

2. The framework

In connection with graph algorithms we use formal languages to describe sets of paths. The letters of the underlying alphabet are interpreted as graph nodes. As a special case of formal languages we consider relations of arities $\leqslant 2$. Relations of arity 1 represent node sets, whereas binary relations represent edge sets. The only two nullary relations (the singleton relation consisting just of the empty word and the

* Corresponding author. E-mail: {moeller,russling}@uni-augsburg.de.

0167-6423/94/$07.00
SSDI 0167-6423(93)E0017-T

empty relation) play the role of the Boolean values. This also allows easy definitions of assertions, conditional, and guards.

Essential operations on languages are (besides union, intersection, and difference) concatenation, composition, and join. As special cases of composition we obtain image and inverse image as well as tests for intersection, emptiness, and membership. The join corresponds to path concatenation on directed graphs; special cases yield restriction.

Proofs are either straightforward or given by Möller [14] and therefore omitted.

2.1. Operations on sets

Given a set A we denote by $\mathscr{P}(A)$ its *powerset*. The cardinality of A is, as usual, denoted by $|A|$. To save braces, we identify a singleton set with its only element.

Frequently, we will extend set-valued operations

$$f: A_1 \times \cdots \times A_n \to \mathscr{P}(A_{n+1}) \quad (n > 0)$$

to the powersets $\mathscr{P}(A_i)$ of the A_i. In these cases we use the same symbol f also for the extended function

$$f: \mathscr{P}(A_1) \times \cdots \times \mathscr{P}(A_n) \to \mathscr{P}(A_{n+1}),$$

defined by

$$f(U_1, \ldots, U_n) \stackrel{\text{def}}{=} \bigcup_{x_1 \in U_1} \cdots \bigcup_{x_n \in U_n} f(x_1, \ldots, x_n) \tag{1}$$

for $U_i \subseteq A_i$. By this definition, the extended operation distributes through union in all arguments:

$$f(U_1, \ldots, U_{i-1}, \bigcup_{j \in J} U_{ij}, U_{i+1}, \ldots, U_n) = \bigcup_{j \in J} f(U_1, \ldots, U_{i-1}, U_{ij}, U_{i+1}, \ldots, U_n). \tag{2}$$

By taking $J = \emptyset$ we obtain strictness of the extended operation w.r.t. $\emptyset$:

$$f(U_1, \ldots, U_{i-1}, \emptyset, U_{i+1}, \ldots, U_n) = \emptyset. \tag{3}$$

By taking $J = \{1, 2\}$ and using the equivalence

$$U \subseteq V \Leftrightarrow U \cup V = V,$$

we also obtain monotonicity w.r.t. $\subseteq$ in all arguments:

$$U_{i1} \subseteq U_{i2} \Rightarrow f(U_1, \ldots, U_{i-1}, U_{i1}, U_{i+1}, \ldots, U_n) \subseteq f(U_1, \ldots, U_{i-1}, U_{i2}, U_{i+1}, \ldots, U_n). \tag{4}$$

Moreover, bilinear equational laws are preserved (see e.g. [11]).

2.2. Languages and relations

Consider an alphabet A. We denote the empty word over A by ε and concatenation by $\bullet$. It is associative, with ε as the neutral element:

$$u \bullet (v \bullet w) = (u \bullet v) \bullet w, \tag{5}$$

$$\varepsilon \bullet u = u = u \bullet \varepsilon. \tag{6}$$

As usual, a singleton word is not distinguished from the only letter it contains. The *length* of a word u, i.e., the number of letters from A in u, is denoted by $\|u\|$. The *reverse* of a word $u = a_1 \bullet \cdots \bullet a_{\|u\|}$ is the word $u^{-1} \stackrel{\text{def}}{=} a_{\|u\|} \bullet \cdots \bullet a_1$. The set of all words over A is denoted by $A^{(*)}$.

A *(formal) language* is a subset of $A^{(*)}$. Concatenation is extended pointwise to languages. Since the above laws are bilinear, they carry over to languages U, V, W over A:

$$U \bullet (V \bullet W) = (U \bullet V) \bullet W, \tag{7}$$

$$\varepsilon \bullet U = U = U \bullet \varepsilon. \tag{8}$$

The *diagonal* V^{Δ} over a subset $V \subseteq A$ is defined by

$$V^{\Delta} \stackrel{\text{def}}{=} \bigcup_{x \in V} x \bullet x. \tag{9}$$

A *relation of arity n* is a language R such that all words in R have length n. Note that $\emptyset$ is a relation of any arity. For $R \neq \emptyset$ we denote the arity of R by $\operatorname{ar} R$. There are only two 0-ary relations, viz. $\emptyset$ and ε. For a relation R its *converse* R^{-1} consists of the reverses of all its words.

2.3. Composition

For languages V and W over alphabet A we define their *composition* $V; W$ by

$$V; W \stackrel{\text{def}}{=} \bigcup_{x \in A} \bigcup_{v \bullet x \in V} \bigcup_{x \bullet w \in W} v \bullet w. \tag{10}$$

If V and W are binary relations this coincides with the usual definition of relational composition (see e.g. [20,21]).

Composition is associative:

$$U;(V; W) = (U; V); W \Leftarrow \forall y \in V: \|y\| \geqslant 2. \tag{11}$$

Composition associates with concatenation:

$$U \bullet (V; W) = (U \bullet V); W \Leftarrow \forall y \in V: \|y\| \geqslant 1, \tag{12}$$

$$U;(V \bullet W) = (U; V) \bullet W \Leftarrow \forall y \in V: \|y\| \geqslant 1. \tag{13}$$

We shall omit parentheses whenever one of these laws applies. Moreover, • and ; bind stronger than $\cup$ and $\cap$.

Interesting special cases of relational composition arise when one of the operands has arity 1. Suppose $1 = \mathsf{ar}\, R \leqslant \mathsf{ar}\, S$. Then

$$R;S = \bigcup_{x \in R} \bigcup_{x \bullet v \in S} v.$$

In other words, $R;S$ is the image of R under S. Likewise, if $1 = \mathsf{ar}\, T \leqslant \mathsf{ar}\, S$, then $S;T$ is the inverse image of T under S. For these reasons we may define domain and codomain of a binary relation R by

$$\mathsf{dom}\, R \stackrel{\text{def}}{=} R;A, \tag{14}$$

$$\mathsf{cod}\, R \stackrel{\text{def}}{=} A;R. \tag{15}$$

Suppose now $\mathsf{ar}\, R = 1 = \mathsf{ar}\, S$ and $\|x\| = 1 = \|y\|$, then

$$R;S = \begin{cases} \varepsilon & \text{if } R \cap S \neq \emptyset, \\ \emptyset & \text{if } R \cap S = \emptyset, \end{cases} \tag{16}$$

$$R;R = \begin{cases} \varepsilon & \text{if } R \neq \emptyset, \\ \emptyset & \text{if } R = \emptyset, \end{cases} \tag{17}$$

$$x;R = R;x = \begin{cases} \varepsilon & \text{if } x \in R, \\ \emptyset & \text{if } x \notin R, \end{cases} \tag{18}$$

$$x;y = y;x = \begin{cases} \varepsilon & \text{if } x = y, \\ \emptyset & \text{if } x \neq y. \end{cases} \tag{19}$$

Because these "tests" will be used frequently, we introduce more readable notations for them by setting

$$(R \neq \emptyset) \stackrel{\text{def}}{=} R;R, \tag{20}$$

$$x \in R \stackrel{\text{def}}{=} x;R, \tag{21}$$

$$(x = y) \stackrel{\text{def}}{=} x;y, \tag{22}$$

$$R \subseteq S \stackrel{\text{def}}{=} (R \cup S = S). \tag{23}$$

For binary R and $x \in \mathsf{dom}\, R$, $y \in \mathsf{cod}\, R$ we have

$$x;R;y = \begin{cases} \varepsilon & \text{if } x \bullet y \in R, \\ \emptyset & \text{otherwise}. \end{cases} \tag{24}$$

Finally, we note that diagonals are neutral w.r.t. composition. Assume $P \supseteq \mathsf{dom}\, V$ and $Q \supseteq \mathsf{cod}\, V$. Then

$$P^{\Delta};V = V, \tag{25}$$

$$V;Q^{\Delta} = V. \tag{26}$$

2.4. Assertions

As we have just seen, the nullary relations ε and $\emptyset$ characterize the outcomes of certain test operations. More generally, they can be used instead of Boolean values; therefore, we call expressions yielding nullary relations *assertions*. Note that in this view "false" and "undefined" both are represented by $\emptyset$. Negation is defined by

$$\bar{\emptyset} \stackrel{\text{def}}{=} \varepsilon, \tag{27}$$

$$\bar{\varepsilon} \stackrel{\text{def}}{=} \emptyset. \tag{28}$$

Note that this operation is not monotonic.

For assertions B and C we have e.g. the properties

$$B \bullet C = B \cap C, \tag{29}$$

$$B \bullet B = B, \tag{30}$$

$$B \bullet \bar{B} = \emptyset, \tag{31}$$

$$B \cup \bar{B} = \varepsilon, \tag{32}$$

$$\overline{B \bullet C} = \bar{B} \cup \bar{C}. \tag{33}$$

Conjunction and disjunction of assertions are represented by their intersection and union. To improve readability, we write $B \wedge C$ for $B \cap C = B \bullet C$ and $B \vee C$ for $B \cup C$.

For assertion B and arbitrary language R we have

$$B \bullet R = R \bullet B = \begin{cases} R & \text{if } B = \varepsilon, \\ \emptyset & \text{if } B = \emptyset. \end{cases} \tag{34}$$

Hence, $B \bullet R$ (and $R \bullet B$) behaves like the expression

$$B \rhd R = \text{if } B \text{ then } R \text{ else error fi}$$

in [13]. We will use this construct for propagating assertions through recursions.

2.5. Conditional

Using assertions we can also define a conditional by

$$\text{if } B \text{ then } R \text{ else } S \text{ fi} \stackrel{\text{def}}{=} B \bullet R \cup \bar{B} \bullet S \tag{35}$$

for assertion B and languages R and S. Note that this operation is not monotonic in B.

2.6. Join

A useful derived operation is provided by a special case of the join operation as used in database theory (see e.g. [8]). Given two languages R and S, their *join* $R \bowtie S$

consists of all words that arise from "glueing" together words from R and from S along a common intermediate letter. By our previous considerations, the beginnings of words ending with $x \in A$ are obtained as $R;x$, whereas the ends of words which start with x are obtained as $x;S$. Hence, we define

$$R \bowtie S \stackrel{\text{def}}{=} \bigcup_{x \in A} R;x \bullet x;S. \tag{36}$$

Again, $\bowtie$ binds stronger than $\cup$ and $\cap$.

Join and composition are closely related. To explain this we consider two binary relations $R, S \subseteq A \bullet A$:

$$R;S = \bigcup_{z \in A} \{x \bullet y\colon x \bullet z \in R \wedge z \bullet y \in S\},$$

$$R \bowtie S = \bigcup_{z \in A} \{x \bullet z \bullet y\colon x \bullet z \in R \wedge z \bullet y \in S\}.$$

Thus, whereas $R;S$ just states whether there is a path from x to y via some point $z \in A$, the relation $R \bowtie S$ consists of exactly those paths $x \bullet z \bullet y$. In particular, the relations

$$R,$$

$$R \bowtie R,$$

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

$$\vdots$$

consist of the paths of edge numbers $1, 2, 3, \ldots$ in the directed graph associated with R.

Other interesting special cases arise when the join is taken w.r.t. the minimum of the arities involved. Suppose $1 = \mathsf{ar}\, R \leqslant \mathsf{ar}\, S$. Then

$$R \bowtie S = \bigcup_{x \in A} R;x \bullet x;S = \bigcup_{x \in R} x \bullet x;S.$$

In other words, $R \bowtie S$ is the restriction of S to R. Likewise, for T with $1 = \mathsf{ar}\, T \leqslant \mathsf{ar}\, S$, the language $S \bowtie T$ is the corestriction of S to T.

If even $\mathsf{ar}\, R = \mathsf{ar}\, S = 1$ we have

$$R \bowtie S = R \cap S. \tag{37}$$

In particular, if $\mathsf{ar}\, R = 1$ and $\|x\| = 1 = \|y\|$,

$$R \bowtie R = R, \tag{38}$$

$$x \bowtie R = R \bowtie x = \begin{cases} x & \text{if } x \in R, \\ \emptyset & \text{if } x \notin R, \end{cases} \tag{39}$$

$$x \bowtie y = y \bowtie x = \begin{cases} x & \text{if } x = y, \\ \emptyset & \text{if } x \neq y. \end{cases} \tag{40}$$

For binary R, $x \in \mathsf{dom}\, R$ and $y \in \mathsf{cod}\, R$, this implies

$$x \bowtie R \bowtie y = \begin{cases} x \bullet y & \text{if } x \bullet y \in R, \\ \emptyset & \text{otherwise.} \end{cases} \tag{41}$$

In special cases, the join can be expressed by a composition: assume $\mathsf{ar}\, P = 1 = \mathsf{ar}\, Q$. Then

$$P \bowtie R = P^A; R, \tag{42}$$

$$R \bowtie Q = R; Q^A. \tag{43}$$

By the associativity of composition (11) also join and composition associate:

$$(R \bowtie S); T = R \bowtie (S; T), \tag{44}$$

$$R; (S \bowtie T) = (R; S) \bowtie T. \tag{45}$$

provided $\mathsf{ar}\, S \geqslant 2$.

Moreover, also joins associate:

$$R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T. \tag{46}$$

2.7. Kleene algebras and closures

A *Kleene algebra* (see [7]) is a tuple $(S, \Sigma, \circ, 0, 1)$ consisting of a set S, operations $\Sigma : \mathscr{P}(S) \to S$ and $\circ : S \bullet S \to S$ and elements $0, 1 \in S$ such that $(S, \circ, 1)$ is a monoid and

$$\begin{aligned} &\Sigma\emptyset = 0, \\ &\Sigma\{x\} = x && (x \in S), \\ &\Sigma(\cup \mathscr{K}) = \Sigma\{\Sigma K : K \in \mathscr{K}\} && (\mathscr{K} \subseteq \mathscr{P}(S)), \\ &\Sigma(K \circ L) = (\Sigma K) \circ (\Sigma L) && (K, L \in \mathscr{P}(S)), \end{aligned} \tag{47}$$

where in this latter equation $\circ$ is the pointwise extension of the monoid operation. Note that this implies that 0 is a zero with respect to $\circ$ or, in other words, that $\circ$ is strict with respect to 0:

$$0 \circ x = 0 = x \circ 0. \tag{48}$$

The binary version of Σ is

$$x + y \stackrel{\text{def}}{=} \Sigma\{x, y\}, \tag{49}$$

which makes $(S, +, 0)$ a commutative monoid. By our definitions, for an alphabet A,

$$LAN \stackrel{\text{def}}{=} (\mathscr{P}(A^{(*)}), \bigcup, \bullet, \emptyset, \varepsilon),$$

$$REL \stackrel{\text{def}}{=} (\mathscr{P}(A \bullet A), \bigcup, ;, \emptyset, A^A),$$

$$PAT \stackrel{\text{def}}{=} (\mathscr{P}(A^{(*)} \setminus \varepsilon), \bigcup, \bowtie, \emptyset, A),$$

all form Kleene algebras. Given a Kleene algebra one can define a partial order $\leqslant$ by

$$x \leqslant y \overset{\text{def}}{\Leftrightarrow} x + y = y. \tag{50}$$

This makes $(S, \leqslant)$ into a complete lattice. Moreover, $\circ$ is continuous w.r.t. $\leqslant$. In our examples $\leqslant$ coincides with $\subseteq$. One can then define a *closure* operator .* by

$$x^* \overset{\text{def}}{=} \mu y.\, 1 + x \circ y, \tag{51}$$

where μ is the least fixpoint operator. Using continuity we can represent the closure also by Kleene's approximation sequence (see [10]) as

$$x^* = \Sigma\{x^j : j \in \mathbb{N}\}, \tag{52}$$

where

$$x^0 \overset{\text{def}}{=} 1, \tag{53}$$

$$x^{j+1} \overset{\text{def}}{=} x \circ x^j. \tag{54}$$

For our particular Kleene algebras we denote the closure operations by $.^{(*)}$, .*, and $.^{\Rightarrow}$, respectively.

Consider now a binary relation $R \subseteq A \bullet A$ and let G be the directed graph associated with R, i.e., the graph with vertex set A and arcs between the vertices corresponding to the pairs in R. We have, in *REL*,

$$x; R^i; y = \begin{cases} \varepsilon & \text{if there is a path with } i \text{ edges from } x \text{ to } y \text{ in } G, \\ \emptyset & \text{otherwise.} \end{cases} \tag{55}$$

Hence,

$$x; R^*; y = \begin{cases} \varepsilon & \text{if there is a path from } x \text{ to } y \text{ in } G, \\ \emptyset & \text{otherwise.} \end{cases} \tag{56}$$

For $S \subseteq A$, the set $S; R^*$ gives all points in A reachable from points in S via paths in G, whereas $R^*; S$ gives all points in A from which some point in S can be reached. Finally,

$$S; R^*; T = \begin{cases} \varepsilon & \text{if } S \text{ and } T \text{ are connected by some path in } G, \\ \emptyset & \text{otherwise.} \end{cases} \tag{57}$$

As usual, we set

$$R^+ \overset{\text{def}}{=} R; R^* = R^*; R. \tag{58}$$

Analogously, the path closure $R^{\Rightarrow}$ in *PAT* consists of all finite paths in G. Hence,

$$x \bowtie R^{\Rightarrow} \bowtie y \tag{59}$$

is the language of all paths between x and y in G.

Moving away from the graph view, the path closure is also useful for general binary relations. Let e.g. $\leqslant$ be a partial order. Then $\leqslant^{\Rightarrow}$ is the language of all $\leqslant$-nondecreasing sequences. If $\leqslant$ is even a linear order then $\leqslant^{\Rightarrow}$ is the language of all sequences which are sorted w.r.t. $\leqslant$. This is exploited in [16,19] for the derivation of sorting algorithms.

We now state some important induction principles for closures. We call a predicate P over a Kleene algebra $(S, \Sigma, \circ, 0, 1)$ *continuous* if for all $T \subseteq S$

$$\left(\bigwedge_{x \in T} P[x] \right) \Rightarrow P[\Sigma T]. \tag{60}$$

Lemma 2.1. *Consider a fixed $z \in S$ and let P be continuous. If $P[1]$ and $P[x] \Rightarrow P[z \circ x]$ or $P[x] \Rightarrow P[x \circ z]$ then $P[z^*]$ holds as well.*

Proof. A straightforward induction shows $P[z^i]$ for all $i \in \mathbb{N}$. Now Kleene's approximation (52) and continuity show the claim. □

Corollary 2.2. *Consider fixed $U \in \mathcal{P}(A^{(*)})$, $R \subseteq A \bullet A$ and suppose that P is a continuous predicate on $\mathcal{P}(A^{(*)})$. If $P[U]$ and $P[V] \Rightarrow P[V;R]$ then $P[U;R^*]$ as well.*

Proof. Define $Q[X]$ over $\mathcal{P}(A \bullet A)$ by

$$Q[X] \overset{\text{def}}{\Leftrightarrow} P[U;X].$$

Then Q satisfies the assumptions of Lemma 2.1 showing the claim. □

A variant of the general induction principle of Lemma 2.1 allows us to extend properties of x to x^*.

Lemma 2.3. *Let P be continuous and assume $P[1]$ and $P[x] \wedge P[y] \Rightarrow P[x \circ y]$. Then $P[x] \Rightarrow P[x^*]$ holds as well.*

Proof. Analogous to the proof of Lemma 2.1. □

We shall see various applications of these principles later on.

3. Graph algorithms

We now want to apply the framework in case studies of some simple graph algorithms.

3.1. Length of a shortest connecting path

3.1.1. Specification and first recursive solution

We consider a finite set A of vertices and a binary relation $R \subseteq A \bullet A$. The problem is to find the length of a shortest path from a vertex x to a vertex y. Therefore, we define

$$shortestpath(x, y) \stackrel{\text{def}}{=} min(edgelengths(x \rtimes R^{\Rightarrow} \rtimes y)), \tag{61}$$

where, for a set S of (nonempty) paths,

$$edgelengths(S) \stackrel{\text{def}}{=} \bigcup_{s \in S} (\|s\| - 1) \tag{62}$$

calculates the set of path lengths, i.e., the number of edges in each path, and, for a set N of natural numbers.

$$min(N) \stackrel{\text{def}}{=} \begin{cases} k & \text{if } k \in N \wedge N \subseteq k; \leqslant, \\ \emptyset & \text{if } N = \emptyset. \end{cases} \tag{63}$$

It is obvious that *edgelengths* is strict and distributes through union. Moreover, for unary S,

$$edgelengths(S \rtimes T) = 1 + edgelengths(S; T), \tag{64}$$

and, for $M, N \subseteq \mathbb{N}$,

$$min(M \cup N) = min(min(M) \cup min(N)), \tag{65}$$

$$min(0 \cup M) = 0. \tag{66}$$

For deriving a recursive version of *shortestpath* we generalize it to a function *sp* which calculates the length of a shortest path from a set S of vertices to a vertex y:

$$sp(S, y) \stackrel{\text{def}}{=} min(edgelengths(S \rtimes R^{\Rightarrow} \rtimes y)). \tag{67}$$

The embedding

$$shortestpath(x, y) = sp(x, y) \tag{68}$$

is straightforward.

We calculate

$sp(S, y)$

$=$ {definition}

$min(edgelengths(S \rtimes R^{\Rightarrow} \rtimes y))$

$=$ {by (51)}

$min(edgelengths(S \rtimes (A \cup R \rtimes R^{\Rightarrow}) \rtimes y))$

$=$ {distributivity}

$$
\begin{aligned}
& min(edgelengths(S \bowtie A \bowtie y) \cup edgelengths(S \bowtie R \bowtie R^{\Rightarrow} \bowtie y)) \\
= \quad & \{\!|\text{by (37)}|\!\} \\
& min(edgelengths(S \bowtie y) \cup edgelengths(S \bowtie R \bowtie R^{\Rightarrow} \bowtie y)).
\end{aligned}
$$

By (39) the subexpression $S \bowtie y$ can be simplified according to whether $y \in S$ or not.

Case 1: $y \in S$.

$$
\begin{aligned}
& min(edgelengths(S \bowtie y) \cup edgelengths(S \bowtie R \bowtie R^{\Rightarrow} \bowtie y)) \\
= \quad & \{\!|\text{by (39), since } y \in S|\!\} \\
& min(edgelengths(y) \cup edgelengths(S \bowtie R \bowtie R^{\Rightarrow} \bowtie y)) \\
= \quad & \{\!|\text{definition of } edgelengths|\!\} \\
& min(0 \cup edgelengths(S \bowtie R \bowtie R^{\Rightarrow} \bowtie y)) \\
= \quad & \{\!|\text{by (66)}|\!\} \\
& 0.
\end{aligned}
$$

Case 2: $y \notin S$.

$$
\begin{aligned}
& min(edgelengths(S \bowtie y) \cup edgelengths(S \bowtie R \bowtie R^{\Rightarrow} \bowtie y)) \\
= \quad & \{\!|\text{by (39), since } y \notin S|\!\} \\
& min(edgelengths(\emptyset) \cup edgelengths(S \bowtie R \bowtie R^{\Rightarrow} \bowtie y)) \\
= \quad & \{\!|\text{strictness, neutrality}|\!\} \\
& min(edgelengths(S \bowtie R \bowtie R^{\Rightarrow} \bowtie y)) \\
= \quad & \{\!|\text{by (64)}|\!\} \\
& min(1 + edgelengths(S ; R \bowtie R^{\Rightarrow} \bowtie y)) \\
= \quad & \{\!|\text{distributivity}|\!\} \\
& 1 + min(edgelengths(S ; R \bowtie R^{\Rightarrow} \bowtie y)) \\
= \quad & \{\!|\text{definition}|\!\} \\
& 1 + sp(S ; R, y).
\end{aligned}
$$

Altogether we have derived the recursion equation

$$sp(S, y) = \text{if } y \in S \text{ then } 0 \text{ else } 1 + sp(S ; R, y) \text{ fi}. \tag{69}$$

Note, however, that termination cannot be guaranteed for this recursion. To make progress in that direction we show some additional properties of *sp*.

Lemma 3.1. $sp(S \cup T, y) = min(sp(S, y) \cup sp(T, y))$.

Proof.

$$
\begin{aligned}
& sp(S \cup T, y) \\
= \; & \{\!\{\text{definition}\}\!\} \\
& min(edgelengths((S \cup T) \bowtie R^{\Rightarrow} \bowtie y)) \\
= \; & \{\!\{\text{distributivity}\}\!\} \\
& min(edgelengths(S \bowtie R^{\Rightarrow} \bowtie y) \cup edgelengths(T \bowtie R^{\Rightarrow} \bowtie y)) \\
= \; & \{\!\{\text{by (65)}\}\!\} \\
& min(min(edgelengths(S \bowtie R^{\Rightarrow} \bowtie y)) \cup min(edgelengths(T \bowtie R^{\Rightarrow} \bowtie y))) \\
= \; & \{\!\{\text{definition}\}\!\} \\
& min(sp(S, y) \cup sp(T, y)). \qquad \square
\end{aligned}
$$

We now consider again the case $y \notin S$. From (69) we obtain

$$sp(S; R, y) \leqslant sp(S, y), \tag{70}$$

and hence

$$
\begin{aligned}
& sp(S, y) \\
= \; & \{\!\{\text{by } y \notin S \text{ and (69)}\}\!\} \\
& 1 + sp(S; R, y) \\
= \; & \{\!\{\text{by (70)}\}\!\} \\
& 1 + min(sp(S, y) \cup sp(S; R, y)) \\
= \; & \{\!\{\text{by Lemma 3.1}\}\!\} \\
& 1 + sp(S \cup S; R, y).
\end{aligned}
$$

so that a second recursion equation for sp is

$$sp(S, y) = \text{if } y \in S \text{ then } 0 \text{ else } 1 + sp(S \cup S; R, y) \text{ fi}. \tag{71}$$

Now, although the first parameter is nondecreasing in each recursive call, still nontermination is guaranteed if there is no path from S to y. However, in that case by finiteness of A the recursive calls of sp eventually become stationary, i.e., eventually $S = S \cup S; R$ holds, which is equivalent to $S; R \subseteq S$. We consider that case in the following lemma.

Lemma 3.2. *If $y \notin S$ and $S; R \subseteq S$ then $S \bowtie R^{\Rightarrow} \bowtie y = \emptyset$, i.e., there is no path from set S to vertex y, and therefore $sp(S, y) = \emptyset$.*

Proof. We use the induction principle of Lemma 2.1 with the predicate

$$P[X] \overset{\text{def}}{\Leftrightarrow} S \rtimes X \rtimes y \subseteq \emptyset .$$

To show $P[A]$ we calculate

$$\begin{aligned}
& S \rtimes A \rtimes y \\
= \ & \{\!\!\{\text{neutrality of } A\}\!\!\} \\
& S \rtimes y \\
= \ & \{\!\!\{\text{by (39), since } y \notin S\}\!\!\} \\
& \emptyset .
\end{aligned}$$

Now assume $P[X]$. We calculate

$$\begin{aligned}
& S \rtimes R \rtimes X \rtimes y \\
= \ & \{\!\!\{\text{by (42)}\}\!\!\} \\
& S^{\Delta}; R \rtimes X \rtimes y \\
\subseteq \ & \{\!\!\{\text{by } S^{\Delta} \subseteq S \bullet S \text{ and monotonicity}\}\!\!\} \\
& S \bullet S; R \rtimes X \rtimes y \\
\subseteq \ & \{\!\!\{\text{by } S; R \subseteq S \text{ and monotonicity}\}\!\!\} \\
& S \bullet S \rtimes X \rtimes y \\
\subseteq \ & \{\!\!\{\text{by } P[X] \text{ and monotonicity}\}\!\!\} \\
& S \bullet \emptyset \\
= \ & \{\!\!\{\text{strictness}\}\!\!\} \\
& \emptyset .
\end{aligned}$$

Now the claim is immediate from the definition of *sp*. □

Altogether we have

$$\begin{aligned}
& \mathit{shortestpath}(x, y) = \mathit{sp}(x, y), \\
& \mathit{sp}(S, y) = \mathsf{if}\ y \in S\ \mathsf{then}\ 0 \\
& \qquad\qquad \mathsf{else\ if}\ S; R \subseteq S\ \mathsf{then}\ \emptyset \\
& \qquad\qquad\qquad\qquad \mathsf{else}\ 1 + \mathit{sp}(S \cup S; R, y)\ \mathsf{fi\ fi}.
\end{aligned} \tag{72}$$

Now termination is guaranteed, since S increases for each recursive call and is bounded by the finite set A of all vertices.

3.1.2. Improving efficiency

One may argue that in the above version, accumulating vertices in the parameter S is not efficient because it makes calculating $S;R$ more expensive. So, in an improved version of the algorithm, we shall keep as few vertices as possible in the parameter S and the set of vertices already visited in an additional parameter T, tied to S by an assertion. Let

$$sp2(S,T,y) \stackrel{\text{def}}{=} (S \cap T = \emptyset \wedge y \notin T) \bullet sp(S \cup T, y), \tag{73}$$

with the embedding

$$shortestpath(x,y) = sp2(x,\emptyset,y). \tag{74}$$

Now assume $S \cap T = \emptyset \wedge y \notin T$. Again we distinguish two cases.

Case 1: $y \in S$.

$sp2(S,T,y)$

$=$ {definition}

$sp(S \cup T, y)$

$=$ {by $y \in S \subseteq S \cup T$ and (72)}

0.

Case 2: $y \notin S$.

$sp2(S,T,y)$

$=$ {definition}

$sp(S \cup T, y)$

$=$ {by $y \notin S \cup T$ and (72)}

if $(S \cup T);R \subseteq S \cup T$ then $\emptyset$

else $1 + sp(S \cup T \cup (S \cup T);R, y)$ fi

$=$ {set theory}

if $(S \cup T);R \subseteq S \cup T$ then $\emptyset$

else $1 + sp(((S \cup T);R)\backslash(S \cup T) \cup (S \cup T), y)$ fi

$=$ {definition and $y \notin S \cup T$}

if $(S \cup T);R \subseteq S \cup T$ then $\emptyset$

else $1 + sp2(((S \cup T);R)\backslash(S \cup T), S \cup T, y)$ fi.

Altogether,

$$shortestpath(x, y) = sp2(x, \emptyset, y),$$

$$sp2(S, T, y) = (S \cap T = \emptyset \wedge y \notin T) \bullet$$

if $y \in S$

then 0

else if $(S \cup T); R \subseteq S \cup T$

then $\emptyset$

else $1 + sp2(((S \cup T); R) \setminus (S \cup T), S \cup T, y)$ fi fi.

This version is still very inefficient. However, a simple analysis shows that the assertion of *sp2* can be strengthened by the conjunct $T; R \subseteq S \cup T$. Thus, one can simplify the program to

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

$$sp3(S, T, y) = (S \cap T = \emptyset \wedge y \notin T \wedge T; R \subseteq S \cup T) \bullet$$

if $y \in S$

then 0

else if $S; R \subseteq S \cup T$

then $\emptyset$

else $1 + sp3((S; R) \setminus (S \cup T), S \cup T, y)$ fi fi.

The formal derivations steps for this are similar to the ones above and hence we omit them.

Termination is guaranteed, since T increases for each recursive call and is bounded by the finite set A of all vertices.

Note that a tail-recursive variant can easily be derived from *sp3* by introducing an accumulator. A corresponding algorithm in iterative form can be found in the literature, e.g. in [9] (but there unfortunately not faultless).

Further, our algorithm also solves the problem whether a vertex y is reachable from a vertex x, since

$$reachable(x, y) = (shortestpath(x, y) \neq \emptyset). \tag{75}$$

3.2. Cycle detection

3.2.1. Problem statement and first solution

Consider again a finite set A of vertices and a binary relation $R \subseteq A \bullet A$. The problem consists in determining whether R contains a *cyclic path*, i.e. a path in which a node occurs twice.

Lemma 3.3. *The following statements are equivalent:*

(1) R *contains a cyclic path.*
(2) $R^+ \cap A^A \neq \emptyset$.
(3) $R^{|A|} \neq \emptyset$.
(4) $R^{|A|}; A \neq \emptyset$.
(5) $A; R^{|A|} \neq \emptyset$.

Proof. (1) $\Rightarrow$(2): Let $p = u \bullet x \bullet v \bullet x \bullet w$ with $x \in A$ and $u, v, w \in A^{(*)}$ be a cyclic path. Then $x \bullet x \in R^+$ and the claim follows.

(2) $\Rightarrow$(3): Assume $x \bullet x \in R^+$ and let n be the smallest number such that there are $x_0, \ldots, x_n \in A$ with

$$\bigcup_{i=0}^{n-1} x_i \bullet x_{i+1} \subseteq R \quad \text{and} \quad x_0 = x = x_n.$$

Then

$$\bullet_{i=0}^{|A|} x_{i \bmod n}$$

is a path as well and hence the claim holds.

(3) $\Rightarrow$(4): Trivial, since $R^{|A|}; A$ is the domain of $R^{|A|}$.

(4) $\Rightarrow$(5): Trivial, since a relation with nonempty domain also has a nonempty codomain.

(5) $\Rightarrow$(1): We have $y \in A; R^{|A|}$ iff there is an $x \in A$ and a path from x to y with $|A| + 1$ nodes. By the pigeonhole principle this path must contain at least one node twice and hence is cyclic. □

By (5) we may specify our problem as

$$\mathit{hascycle} \stackrel{\text{def}}{=} (A; R^{|A|} \neq \emptyset).$$

To compute $A; R^{|A|}$ we define $A_i \stackrel{\text{def}}{=} A; R^i$ and use the properties of the powers of R:

$$A_0 = A; R^0 = A; A^A = A,$$

$$A_{i+1} = A; R^{i+1} = A; (R^i; R) = (A; R^i); R = A_i; R.$$

The associated function

$$f: X \mapsto X; R$$

is monotonic. We now prove a general theorem about monotonic functions on noetherian partial orders. A partial order $(M, \leqslant)$ is *noetherian* if each of its nonempty subsets has a minimal element w.r.t. $\leqslant$. An element $x \in S \subseteq M$ is *minimal* in S if $y \in S$ and $y \leqslant x$ imply $y = x$. Using reflexivity of $\leqslant$ we therefore have that x is minimal in S iff

$$\leqslant; x \cap S = x. \tag{76}$$

Viewing a function $f: M \to M$ as a binary relation, we can form its closure f^*. Then, for $x \in M$, we have

$$x; f^* = \{f^i(x): i \in \mathbb{N}\}, \tag{77}$$

where the f^i are defined as usual.

Theorem 3.4. *Let $(M, \leqslant)$ be a noetherian partial order and $f: M \to M$ a monotonic total function.*

(1) *If for $x \in M$ we have $f(x) \leqslant x$ then $x_\infty \stackrel{\text{def}}{=} \mathsf{glb}\,(x; f^*)$ exists and is a fixpoint of f.*
(2) *Assume x as in* (1) *and $y \in M$ with $x_\infty \leqslant y \leqslant x$. Then also $y_\infty = \mathsf{glb}(y; f^*)$ exists and $y_\infty = x_\infty$.*
(3) *If M has a greatest element T then T_∞ exists and is the greatest fixpoint of f.*

Proof. We first restate in relational notation some of the notions involved.

(a) f is a total function iff $A^\Delta \subseteq f; f^{-1}$ and $f^{-1}; f \subseteq A^\Delta$.
(b) f is monotonic iff $\leqslant; f \subseteq f; \leqslant$.
(c) $u \leqslant v$ is equivalent to both $u \subseteq \leqslant; v$ and $v \subseteq u; \leqslant$.
(d) By our convention about singleton sets, $f(x)$ and $x; f$ are the same for a total function f.

In particular, (a) implies, for $S \subseteq M$.

$$(S; f)^\Delta = f^{-1}; S^\Delta; f. \tag{78}$$

We now treat the claims in order.

(1) We first show that

$$\forall y \in x; f^*: f(y) \leqslant y. \tag{79}$$

Relationally, this means $(x; f^*)^\Delta \subseteq f; \leqslant$. We use the induction principle from Corollary 2.2 with the predicate

$$P[X] \stackrel{\text{def}}{\Leftrightarrow} X^\Delta \subseteq f; \leqslant$$

and $U = x$, $R = f$. $P[x]$ holds by assumption. To infer $P[X; f]$ from $P[X]$ we calculate

$$\begin{aligned}
&(X; f)^\Delta \\
=\ &\{\!\!\{\text{by (78)}\}\!\!\} \\
&f^{-1}; X^\Delta; f \\
\subseteq\ &\{\!\!\{\text{by } P[X] \text{ and monotonicity}\}\!\!\} \\
&f^{-1}; f; \leqslant; f \\
\subseteq\ &\{\!\!\{f \text{ is a function, neutrality}\}\!\!\}
\end{aligned}$$

$$
\begin{array}{ll}
 & \leqslant ; f \\
\subseteq & \{\!| f \text{ monotonic} |\!\} \\
 & f ; \leqslant .
\end{array}
$$

Since $(M, \leqslant)$ is noetherian, $x ; f^*$ has a minimal element x_∞. For this we calculate

$$
\begin{array}{ll}
 & x_\infty \\
= & \{\!| \text{by minimality (76)} |\!\} \\
 & \leqslant ; x_\infty \cap x ; f^* \\
\supseteq & \{\!| \text{by (79) and monotonicity of } \cap |\!\} \\
 & x_\infty ; f \cap x ; f^* \\
= & \{\!| \text{since } x_\infty ; f \subseteq x ; f^* |\!\} \\
 & x_\infty ; f .
\end{array}
$$

However, by totality of f this means $x_\infty ; f = x_\infty$ so that x_∞ is a fixpoint of f. We shall see below that $x_\infty = \mathsf{glb}(x ; f^*)$.

(2) Using the induction principle from Lemma 2.3 it is immediate to show the following facts:

$$X ; f \subseteq X \;\Rightarrow\; X ; f^* \subseteq X , \tag{80}$$

$$\leqslant ; f \subseteq f ; \leqslant \;\Rightarrow\; \leqslant ; f^* \subseteq f^* ; \leqslant , \tag{81}$$

$$x ; f \subseteq \leqslant ; x \;\Rightarrow\; x ; f^* \subseteq \leqslant ; x . \tag{82}$$

From (80) and $X \subseteq X ; f^*$ by $A^A \subseteq f^*$ we infer

$$X ; f \subseteq X \;\Rightarrow\; X ; f^* = X . \tag{83}$$

Now consider $u, v \in M$ with $u \leqslant v$. Then

$$
\begin{array}{ll}
 & u \leqslant v \\
\Leftrightarrow & \{\!| \text{image} |\!\} \\
 & v \subseteq u ; \leqslant \\
\Rightarrow & \{\!| \text{monotonicity} |\!\} \\
 & v ; f^* \subseteq u ; \leqslant ; f^* \\
\Rightarrow & \{\!| f \text{ monotonic and (81)} |\!\} \\
 & v ; f^* \subseteq u ; f^* ; \leqslant ,
\end{array}
$$

so that $u;f^*$ is a minorant for $v;f^*$. In particular, taking $u = x_\infty$ and $v = x$ or $v = y$ and using (83) and (82) we see that x_∞ is a lower bound for both $y;f^*$ and $x;f^*$. This also implies $x_\infty = \mathsf{glb}(x;f^*)$. Moreover, $y;f^*$ is a minorant of $x;f^*$ so that every lower bound z for $y;f^*$ also is a lower bound for $x;f^*$ and hence satisfies $z \leqslant x_\infty$. But this shows $x_\infty = \mathsf{glb}(y;f^*) = y_\infty$.

(3) Trivially, $f(\mathsf{T}) \leqslant \mathsf{T}$, and hence T_∞ exists by (1). Let x be a fixpoint of f. By the proof of (2) we know that $\mathsf{T};f^* \subseteq x;f^*;\ \leqslant\ = x;\leqslant$ so that x is a lower bound for $\mathsf{T};f^*$. This implies $x \leqslant \mathsf{glb}(\mathsf{T};f^*) = \mathsf{T}_\infty$. □

A similar theorem has been stated by Cai and Paige [6].

Corollary 3.5. *If* $\mathsf{T}_\infty \leqslant x$ *for some* $x \in M$ *then* $\mathsf{T}_\infty = x_\infty$.

Proof. By (2) of the above theorem. □

To calculate x_∞ we define a function *inf* by

$$inf(y) \stackrel{\text{def}}{=} (x_\infty \leqslant y \leqslant x)\bullet\, x_\infty\,,$$

which determines x_∞ using an upper bound y. We have the embedding $x_\infty = inf(x)$. Now from the proof of the above theorem the following recursion is immediate:

$$inf(y) = (x_\infty \leqslant y \leqslant x)\bullet \text{ if } y = f(y) \text{ then } y \text{ else } inf(f(y)) \text{ fi}\,.$$

This recursion terminates for every y satisfying $f(y) \leqslant y$, since monotonicity then also shows $f(f(y)) \leqslant f(y)$, so that in each recursive call the parameter decreases properly. In particular, the call $inf(x)$ terminates. This algorithm is an abstraction of many iteration methods on finite sets.

We now return to the special case of cycle detection. By finiteness of A the partial order $(\mathscr{P}(A), \subseteq)$ is noetherian with greatest element A. Therefore, A_∞ exists. Moreover, we have the following corollary.

Corollary 3.6. $A_{|A|} = A_\infty$.

Proof. The length of any properly descending chain in $\mathscr{P}(A)$ is at most $|A| + 1$. Hence, we have $A_{|A|+1} = A_{|A|}$ and thus $A_{|A|} = A_\infty$. □

So we have reduced our task to checking whether $A_\infty \neq \emptyset$, i.e., whether $inf(A) \neq \emptyset$. For our special case the recursion for *inf* reads (omitting the trivial part $W \subseteq A$)

$$inf(W) = (A_\infty \subseteq W)\bullet$$
$$\text{if } W = W;R \text{ then } W \text{ else } inf(W;R) \text{ fi}\,.$$

We want to improve this by avoiding the computation of $W;R$. By the above considerations we may strengthen the assertion of *inf* by adding the conjunct

$W;R \subseteq W$. We define

$$src(W) \stackrel{\text{def}}{=} W \backslash (W;R).$$

This is the set of *sources* of W, i.e., the set of nodes in W which do not have a predecessor in W.

Now, assuming $W;R \subseteq W$, we have $W = W;R \Leftrightarrow src(W) = \emptyset$ and $W;R = W \backslash src(W)$ so that we can rewrite inf into

$$inf(W) = (A_\infty \subseteq W \wedge W;R \subseteq W) \bullet$$
$$\text{if } src(W) = \emptyset \text{ then } W \text{ else } inf(W \backslash src(W)) \text{ fi}.$$

This is an improvement in that $src(W)$ usually will be small compared to W; moreover, the computation of $src(W)$ can be facilitated by a suitable representation of R.

Plugging this into our original problem of cycle recognition we obtain

$$hascycle = hcy(A), \tag{84}$$

where

$$hcy(W) = (A_\infty \subseteq W \wedge W;R \subseteq W) \bullet$$
$$\text{if } src(W) = \emptyset \text{ then } W \neq \emptyset \text{ else } hcy(W \backslash src(W)) \text{ fi}, \tag{85}$$

which is one of the classical algorithms which works by successive removal of sources (see e.g. [3]). Note that Lemma 3.3(4) suggests a dual specification to the one we have used; replaying our development for it would lead to an algorithm that works by successive removal of sinks.

3.2.2. Improving efficiency

We want to improve the computation of the sets $src(W)$. We observe that

$$\begin{aligned} & x \in src(W) \\ = \; & x \in W \backslash (W;R) \\ = \; & x \in W \wedge x \notin W;R \\ = \; & x \in W \wedge R;x \cap W = \emptyset \\ = \; & x \in W \wedge |R;x \cap W| = 0. \end{aligned}$$

So we define for $W \subseteq A$ the relation $in(W)$ by

$$x;in(W) \stackrel{\text{def}}{=} |R;x \cap W|. \tag{86}$$

Hence, $x;in(W)$ gives the indegree of x w.r.t. W and

$$src(W) = W \cap in(W);0. \tag{87}$$

In final implementation, $in(W)$ will, of course, be realized by an array. We aim at an incremental updating of in in the course of our algorithm. We calculate

$$
\begin{aligned}
& x;in(W\backslash src(W)) \\
= \;& \{\!\{\text{definition}\}\!\} \\
& |R;x \cap (W\backslash src(W))| \\
= \;& \{\!\{\text{set theory}\}\!\} \\
& |(R;x \cap W)\backslash src(W)| \\
= \;& \{\!\{|A\backslash B| = |A| - |A \cap B|\}\!\} \\
& |(R;x \cap W)| - |R;x \cap W \cap src(W)| \\
= \;& \{\!\{src(W) \subseteq W\}\!\} \\
& |(R;x \cap W)| - |R;x \cap src(W)| \\
= \;& \{\!\{\text{definition}\}\!\} \\
& x;in(W) - x;in(src(W)).
\end{aligned}
$$

For binary relations f, g with the same domain and subsets of $\mathbb{N}$ as codomains and arithmetic operator $\wr$ we define $f \wr g$ by

$$x;(f \wr g) \stackrel{\text{def}}{=} (x;f) \wr (x;g). \tag{88}$$

Then

$$in(W\backslash src(W)) = in(W) - in(src(W)). \tag{89}$$

For the computation of *in* we observe that

$$in(\emptyset) = \mathbf{0}, \tag{90}$$

where

$$x;\mathbf{0} \stackrel{\text{def}}{=} 0. \tag{91}$$

Moreover, if $S \neq \emptyset$ and $q \in S$ is arbitrary we have

$$
\begin{aligned}
& x;in(S) \\
= \;& x;in(q \cup S\backslash q) \\
= \;& |R;x \cap (q \cup S\backslash q)| \\
= \;& |(R;x \cap q) \cup (R;x \cap S\backslash q)| \\
= \;& |R;x \cap q| + |R;x \cap S\backslash q| \\
= \;& x;in(q) + x;in(S\backslash q),
\end{aligned}
$$

when

$$x;in(q) = \text{if } q;R;x \text{ then } 1 \text{ else } 0 \text{ fi}. \tag{92}$$

Then

$$in(S) = in(q) + in(S \setminus q). \tag{93}$$

We forego a transformation of *in* into tail recursive form, since this is completely standard using associativity of $+$.

Now we can administer the source sets more efficiently. We introduce additional parameters for carrying along $src(W)$ and $in(W)$ and adjust these parameters by the technique of finite differencing (see e.g. [18]). We set, for $S \subseteq W \subseteq A$ and relation f,

$$hc(W,S,f) \stackrel{\text{def}}{=} (S = src(W) \wedge f = in(W)) \bullet hcy(W)), \tag{94}$$

with the embedding

$$hcy(W) = hc(W, src(W), in(W)). \tag{95}$$

Now

$hc(W,S,f)$

$=$ {definitions}

if $src(W) = \emptyset$ then $W \neq \emptyset$ else $hcy(W \setminus src(W))$

$=$ {assertion}

if $S = \emptyset$ then $W \neq \emptyset$ else $hcy(W \setminus S)$

$=$ {embedding}

if $S = \emptyset$ then $W \neq \emptyset$ else $hc(W \setminus S, src(W \setminus S), in(W \setminus S))$

$=$ {introducing auxiliaries}

if $S = \emptyset$ then $W \neq \emptyset$

else let $T \stackrel{\text{def}}{=} W \setminus S$

let $g \stackrel{\text{def}}{=} in(T)$

in $hc(T, src(T), g)$ fi.

$=$ {by (89) and (87)}

if $S = \emptyset$ then $W \neq \emptyset$

else let $T \stackrel{\text{def}}{=} W \setminus S$

let $g \stackrel{\text{def}}{=} f - in(S)$

in $hc(T, T \cap g; 0, g)$ fi.

A final improvement would consist in merging the computation of g with that of $T \cap g; 0$ using the tupling strategy (see e.g. [18]).

4. Conclusion

The calculus of formal languages and relations has proved to speed up derivations; in particular, the way from "nonoperational" specifications involving the closures R^* and $R^{\overrightarrow{+}}$ to first recursive solutions. However, also the tuning steps in improving the recursions have benefitted from the quantifier-free notation. If the resulting derivations still appear lengthy, this is to a great deal due to the fact that the assertions have been constructed in a stepwise fashion (for mastering complexity) rather than in one blow. Further case studies which demonstrate the viability of the approach in more complicated examples are under way. Moreover, the framework has been applied to other problem areas as well (see [15,17]).

Currently, we are working the definition of a more general program development language based on this approach. While other authors use a purely relational approach employing mostly even only binary relations, we find that relations with their fixed arity are too unflexible and lead to a lot of unnecessary encoding and decoding.

Acknowledgement

We are grateful to H. Partsch and to anonymous referees for a number of valuable remarks.

References

[1] F.L. Bauer, R. Berghammer, M. Broy, W. Dosch, F. Geiselbrechtinger, R. Gnatz, E. Hangel, W. Hesse, B. Krieg-Brückner, A. Laut. T.A. Matzner, B. Möller, F. Nickl, H. Partsch, P. Pepper, K. Samelson, M. Wirsing and H. Wössner, *The Munich Project CIP, Volume I: The Wide Spectrum Language CIP-L*, Lecture Notes in Computer Science **183** (Springer, Berlin, 1985).

[2] F.L. Bauer, B. Möller, H. Partsch and P. Pepper, Formal program construction by transformations—computer-aided, intuition-guided programming, *IEEE Trans. Software Engrg.* **15** (1989) 165–180.

[3] R. Berghammer, A transformational development of several algorithms for testing the existence of cycles in a directed graph, Institut für Informatik der TU München, TUM-I8615, München (1986).

[4] R.S. Bird, Lectures on constructive functional programming, in: M. Broy, ed., *Constructive Methods in Computing Science*, NATO ASI Series Series F **55** (Springer, Berlin, 1989) 151–216.

[5] R.M. Burstall and J. Darlington, A transformation system for developing recursive programs, *J. ACM* **24** (1977) 44–67.

[6] J. Cai and R. Paige, Program derivation by fixed point computation, *Sci. Comput. Programming* **11** (1989) 197–261.

[7] J.H. Conway, *Regular Algebra and Finite Machines* (Chapman and Hall, London, 1971).

[8] C.J. Date, *An Introduction to Database Systems, Vol. I* (Addison-Wesley, Reading, MA, 4th ed., 1988).

[9] M. Gondran and M. Minoux, *Graphes et Algorithmes* (Eyrolles, Paris, 1979).

[10] S.C. Kleene, *Introduction to Metamathematics* (Van Nostrand, New York, 1952).

[11] P. Lescanne, Modèles non déterministes de types abstraits, *R.A.I.R.O. Inform. Théor.* **16** (1982) 225–244.

[12] L.G.L.T. Meertens, Algorithmics—towards programming as a mathematical activity, in: J.W. de Bakker et al., eds., *Proceedings CWI Symposium on Mathematics and Computer Science*, CWI Monographs **1** (North-Holland, Amsterdam, 1986) 289–334.

[13] B. Möller, Applicative assertions, in: J.L.A. van de Snepscheut ed., *Mathematics of Program Construction*, Lecture Notes in Computer Science **375** (Springer, Berlin, 1989) 348–362.

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

[15] B. Möller, Towards pointer algebra, *Sci. Comput. Programming* 21 (1993) 57–90.

[16] B. Möller, Algebraic calculation of graph and sorting algorithms, in: D. Bjørner, M. Broy and I.V. Pottosin, eds., *Formal Methods in Programming and their Applications*, Lecture Notes in Computer Science **735** (Springer, Berlin, 1993) 394–413.

[17] B. Möller, Derivation of graph and pointer algorithms, in: B. Möller, H.A. Partsch and S.A. Schuman, eds., *Formal Program Development*, Lecture Notes in Computer Science **755** (Springer, Berlin, 1993) 123–160.

[18] H.A. Partsch, *Specification and Transformation of Programs—A Formal Approach to Software Development* (Springer, Berlin, 1990).

[19] M. Russling, Hamiltonian sorting, Institut für Mathematik der Universität Augsburg, Report No. 270 (1992).

[20] G. Schmidt and T. Ströhlein, *Relations and Graphs*, Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoretical Computer Science (Springer, Berlin, 1993).

[21] A. Tarski, On the calculus of relations, *J. Symbolic Logic* **6** (1941) 73–89.