Shorter Paths to Graph Algorithms

Bernhard Möller and Martin Russling

Institut für Mathematik, Universität Augsburg,
Universitätsstr. 2, W-8900 Augsburg, Germany,
e-mail: {moeller,russling}@uni-augsburg.de

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 Burstall, Darlington 1977, Bauer et al. 1985, Bauer et al. 1989, Meertens 1986, Bird 1989). In it, one starts from a (possibly non-executable) 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 Möller 1991 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 ≤ 2. Relations of arity 1 represent node sets, whereas binary relations represent the edge sets. The only two nullary relations (the singleton relation consisting just of the empty word and the 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 1991 and therefore omitted.

2.1 Operations on Sets

Given a set A we denote by $\mathcal{P}(A)$ 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

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

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

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

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:

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

Moreover, bilinear equational laws are preserved (see e.g. Lescanne 1982).

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 $||u||$.

A **(formal) language** is a set of words over 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 $\mathsf{ar}\, R$. There are only two 0-ary relations, viz. $\emptyset$ and ε.

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. Tarski 1941, Schmidt, Ströhlein 1989).

Composition is associative:

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

Composition associates with concatenation:

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

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

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

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

$$R \neq \emptyset = R\,;R \ , \tag{20}$$

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

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

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

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

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

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

Note that this operation is not monotonic.

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

$$B \bullet C = B \cap C \ , \tag{29}$$
$$B \bullet B = B \ , \tag{30}$$
$$B \bullet \overline{B} = \emptyset \ , \tag{31}$$
$$B \cup \overline{B} = \varepsilon \ , \tag{32}$$
$$\overline{B \bullet C} = \overline{B} \cup \overline{C} \ . \tag{33}$$

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

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

in Möller 1989. We will use this construct for propagating assertions through recursions.

2.5 Conditional

Using assertions we can also define a conditional by

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

for assertion B and languages R, 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. Date 1988). Given two languages R, 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

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

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

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

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

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

$$\begin{array}{l} R \ , \\ R \bowtie R \ , \\ R \bowtie (R \bowtie R) \ , \\ \quad \vdots \end{array}$$

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 $1 = \mathsf{ar}\, R \leq \mathsf{ar}\, S$. Then

$$\begin{array}{ll} & R \bowtie S \\ = & \bigcup\limits_{x\in A} R\,; x \bullet x \bullet x\,; S \\ = & \bigcup\limits_{x\in R} x \bullet x\,; S \ . \end{array}$$

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

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

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

$$P \bowtie R = P^{\Delta}\,;R \ , \tag{42}$$
$$R \bowtie Q = R\,;Q^{\Delta} \ . \tag{43}$$

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 $\mathsf{ar}\, S \geq 2$.

Moreover, also joins associate:

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

2.7 Closures

Consider a binary relation $R \subseteq A \bullet A$. We define the **(reflexive and transitive) closure** R^* of R by

$$R^* \stackrel{\text{def}}{=} \bigcup_{i \in \mathbb{N}} R^i \ , \tag{47}$$

where, as usual,

$$R^0 \stackrel{\text{def}}{=} A^{\Delta} \ , \tag{48}$$
$$R^{i+1} \stackrel{\text{def}}{=} R\,;R^i \ . \tag{49}$$

It is well-known that R^* is the least fixpoint of the recursion equations

$$R^* \ = \ A^{\Delta} \cup R\,;R^* \ = \ A^{\Delta} \cup R^*\,;R \ . \tag{50}$$

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

$$x\,;R^i\,;y = \begin{cases} \varepsilon \text{ if there is a path with } i \text{ edges from } x \text{ to } y \text{ in } G \ , \\ \emptyset \text{ otherwise} \ . \end{cases} \tag{51}$$

Likewise,

$$x\,;R^*\,;y = \begin{cases} \varepsilon \text{ if there is a path from } x \text{ to } y \text{ in } G \ , \\ \emptyset \text{ otherwise} \ . \end{cases} \tag{52}$$

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,

$$s\,;R^*;t = \begin{cases} \varepsilon \text{ if } s \text{ and } t \text{ are connected by some path in } G \ , \\ \emptyset \text{ otherwise} \ . \end{cases} \tag{53}$$

As usual, we set

$$R^+ \stackrel{\text{def}}{=} R\,;R^* \;=\; R^*\,;R \;. \tag{54}$$

Analogously, we define the **path closure** $R^{\Rightarrow}$ of R by

$$R^{\Rightarrow} \stackrel{\text{def}}{=} \bigcup_{i\in\mathbb{N}} {}^iR \;, \tag{55}$$

where now

$${}^0R \stackrel{\text{def}}{=} A \;, \tag{56}$$

$${}^{i+1}R \stackrel{\text{def}}{=} R \bowtie {}^iR \;. \tag{57}$$

It is the least fixpoint of the recursion equations

$$R^{\Rightarrow} \;=\; A \cup R \bowtie R^{\Rightarrow} \;=\; A \cup R^{\Rightarrow} \bowtie R \;. \tag{58}$$

The path closure 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. Analogously to R^+ we define the **proper path closure** by

$$R^{\rightarrow} \stackrel{\text{def}}{=} R \bowtie R^{\Rightarrow} \;=\; R^{\Rightarrow} \bowtie R \;=\; R^{\Rightarrow}\backslash A \;. \tag{60}$$

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

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

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

where, for a set S of (non-empty) paths,

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

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

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

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

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

and

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

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

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

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

The embedding

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

is straightforward.

We calculate

$$\begin{array}{ll} & sp(S,y) \\ = & \quad \{\![\text{ definition }]\!\} \\ & min(edgelengths(S \bowtie R^{\Rightarrow} \bowtie y)) \\ = & \quad \{\![\text{ by (58) }]\!\} \\ & min(edgelengths(S \bowtie (A \;\cup\; R \bowtie R^{\Rightarrow}) \bowtie y)) \\ = & \quad \{\![\text{ distributivity }]\!\} \\ & min(edgelengths(S \bowtie A \bowtie y) \;\cup\; edgelengths(S \bowtie R \bowtie R^{\Rightarrow} \bowtie y)) \\ = & \quad \{\![\text{ by (37) }]\!\} \\ & min(edgelengths(S \bowtie y) \;\cup\; edgelengths(S \bowtie R \bowtie R^{\Rightarrow} \bowtie y)) \;\;. \end{array}$$

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

Case 1: $y \in S$.

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

Case 2: $y \notin S$.

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

Altogether we have derived the recursion equation

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

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

Lemma 1. $sp(S \cup T \ , \ y) \ = \ min(sp(S, y) \cup sp(T, y))$.

Proof.
$$
\begin{array}{ll}
 & sp(S \cup T \ , \ y) \\
= & \quad \{\![\text{ definition }]\!\} \\
 & min(edgelengths((S \cup T) \bowtie R^{\Rightarrow} \bowtie y)) \\
= & \quad \{\![\text{ distributivity }]\!\} \\
 & min(edgelengths(S \bowtie R^{\Rightarrow} \bowtie y) \ \cup \ edgelengths(T \bowtie R^{\Rightarrow} \bowtie y)) \\
= & \quad \{\![\text{ by (65) }]\!\} \\
 & min(min(edgelengths(S \bowtie R^{\Rightarrow} \bowtie y)) \ \cup \ min(edgelengths(T \bowtie R^{\Rightarrow} \bowtie y))) \\
= & \quad \{\![\text{ definition }]\!\} \\
 & min(sp(S, y) \cup sp(T, y)) \ .
\end{array}
$$

□

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

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

and hence

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

so that a second recursion equation for sp is

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

Now, although the first parameter is non-decreasing 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 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. Using the least fixpoint property of $R^{\Rightarrow}$ we use computational induction (see e.g. Manna 1974) to show $S \bowtie R^{\Rightarrow} \bowtie y \subseteq \emptyset$. We use the predicate

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

The induction base $P[\emptyset]$ is trivial by strictness. For the induction step we have to show $P[X] \Rightarrow P[A \cup R \bowtie X]$. Assume $P[X]$. We calculate

$$
\begin{aligned}
& S \bowtie (A \cup R \bowtie X) \bowtie y \\
= \quad & \{\![\text{ distributivity }]\!\} \\
& S \bowtie A \bowtie y \cup S \bowtie R \bowtie X \bowtie y \\
= \quad & \{\![\text{ by (37) }]\!\} \\
& S \bowtie y \ \cup\ S \bowtie R \bowtie X \bowtie y \\
= \quad & \{\![\text{ by (39), since } y \notin S, \text{ and neutrality }]\!\} \\
& S \bowtie R \bowtie X \bowtie y \\
= \quad & \{\![\text{ by (42) }]\!\} \\
& S^{\Delta}\,;R \bowtie X \bowtie y \\
\subseteq \quad & \{\![\text{ by } S^{\Delta} \subseteq S \bullet S \text{ and monotonicity }]\!\} \\
& S \bullet S\,;R \bowtie X \bowtie y
\end{aligned}
$$

$$
\begin{array}{ll}
\subseteq & \{\![\text{ by } S\,;R \subseteq S \text{ and monotonicity }]\!\} \\
& S \bullet S \bowtie X \bowtie y \\
\subseteq & \{\![\text{ by } P[X] \text{ and monotonicity }]\!\} \\
& S \bullet \emptyset \\
= & \{\![\text{ strictness }]\!\} \\
& \emptyset \ .
\end{array}
$$

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

Altogether we have:

$$
\begin{array}{l}
\mathit{shortestpath}(x,y) \ = \ \mathit{sp}(x,y) \ , \\[2ex]
\mathit{sp}(S,y) \ = \ \mathsf{if}\ y \in S\ \mathsf{then}\ 0 \\
\qquad\qquad\qquad\qquad \mathsf{else}\ \ \mathsf{if}\ S\,;R \subseteq S\ \mathsf{then}\ \emptyset \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \ 1 + \mathit{sp}(S \cup S\,;R\ ,\ y)\ \mathsf{fi}\ \mathsf{fi}\ \ .
\end{array}
\tag{72}
$$

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

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

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

with the embedding

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

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

Case 1: $y \in S$.

$$
\begin{array}{ll}
& \mathit{sp2}(S,T,y) \\
= & \{\![\text{ definition }]\!\} \\
& \mathit{sp}(S \cup T\ ,\ y) \\
= & \{\![\text{ by } y \in S \subseteq S \cup T \text{ and (72) }]\!\} \\
& 0 \ .
\end{array}
$$

Case 2: $y \notin S$.

$$
\begin{array}{ll}
 & sp2(S,T,y) \\
= & \quad \{\![\text{ definition }]\!\} \\
 & sp(S \cup T \ , \ y) \\
= & \quad \{\![\text{ by } y \notin S \cup T \text{ and (72) }]\!\} \\
 & \mathsf{if}\ (S \cup T)\,;R \subseteq S \cup T\ \mathsf{then}\ \emptyset \\
 & \qquad\qquad\qquad\qquad \mathsf{else}\ 1 + sp(S \cup T \cup (S \cup T)\,;R \ , \ y)\ \mathsf{fi} \\
= & \quad \{\![\text{ set theory }]\!\} \\
 & \mathsf{if}\ (S \cup T)\,;R \subseteq S \cup T\ \mathsf{then}\ \emptyset \\
 & \qquad\qquad\qquad\qquad \mathsf{else}\ 1 + sp(((S \cup T)\,;R) \backslash (S \cup T) \ \cup \ (S \cup T) \ , \ y)\ \mathsf{fi} \\
= & \quad \{\![\text{ definition and } y \notin S \cup T \ ]\!\} \\
 & \mathsf{if}\ (S \cup T)\,;R \subseteq S \cup T\ \mathsf{then}\ \emptyset \\
 & \qquad\qquad\qquad\qquad \mathsf{else}\ 1 + sp2(((S \cup T)\,;R) \backslash (S \cup T) \ , \ S \cup T \ , \ y)\ \mathsf{fi}\ .
\end{array}
$$

Altogether,

$$
\begin{array}{l}
shortestpath(x,y) \ = \ sp2(x,\emptyset,y) \ , \\
\\
sp2(S,T,y) \ = \ (S \cap T \ = \ \emptyset \ \wedge \ y \notin T) \bullet \\
\qquad\qquad \mathsf{if}\ y \in S \\
\qquad\qquad \mathsf{then}\ 0 \\
\qquad\qquad \mathsf{else}\ \mathsf{if}\ (S \cup T)\,;R \subseteq S \cup T \\
\qquad\qquad\qquad\qquad \mathsf{then}\ \emptyset \\
\qquad\qquad\qquad\qquad \mathsf{else}\ 1 + sp2(((S \cup T)\,;R) \backslash (S \cup T) \ , \ S \cup T \ , \ y)\ \mathsf{fi}\ \mathsf{fi}\ .
\end{array}
$$

This version still is 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

$$
\begin{array}{l}
shortestpath(x,y) \ = \ sp3(x,\emptyset,y) \ , \\
\\
sp3(S,T,y) \ = \ (S \cap T \ = \ \emptyset \ \wedge \ y \notin T \ \wedge \ T\,;R \subseteq S \cup T) \bullet \\
\qquad\qquad \mathsf{if}\ y \in S \\
\qquad\qquad \mathsf{then}\ 0 \\
\qquad\qquad \mathsf{else}\ \mathsf{if}\ S\,;R \subseteq S \cup T \\
\qquad\qquad\qquad\qquad \mathsf{then}\ \emptyset \\
\qquad\qquad\qquad\qquad \mathsf{else}\ 1 + sp3((S\,;R) \backslash (S \cup T) \ , \ S \cup T \ , \ y)\ \mathsf{fi}\ \mathsf{fi}\ .
\end{array}
$$

The formal derivation 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 Gondran, Minoux 1979 (but there unfortunately not faultless).

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

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

3.2 Cycle Detection

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. *The following statements are equivalent:*
(1) *R contains a cyclic path.*
(2) $R^+ \cap A^\Delta \neq \emptyset$.
(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 $\bigcup_{i=0}^{n-1} x_i \bullet x_{i+1} \subseteq R$ and $x_0 = x = x_n$. 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:

$$\begin{aligned} A_0 \;&=\; A ; R^0 \;=\; A ; A^\Delta \;=\; A \;, \\ A_{i+1} \;&=\; A ; R^{i+1} \;=\; A ; (R^i ; R) \;=\; (A ; R^i) ; R \;=\; A_i ; R \;. \end{aligned}$$

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 $(M, \leq)$ is **noetherian** if every descending sequence in it becomes stationary or, equivalently, if each of its nonempty subsets has a minimal element.

Theorem 4. *Let $(M, \leq)$ be a noetherian partial order and let $f : M \to M$ be monotonic.*
(1) *If for $x \in M$ we have $f(x) \leq x$, then $x_\infty \stackrel{\text{def}}{=} \text{glb}\ \{f^i(x) : i \in \mathbb{N}\}$ exists and is a fixpoint of f.*
(2) *If for $x, y \in M$ we have $f(x) \leq x$ and $x_\infty \leq y \leq x$, then also y_∞ exists and $x_\infty = y_\infty$.*
(3) *If M has a greatest element $\top$ then $\top_\infty$ is the greatest fixpoint of f.*

Proof. (1) By assumption we have $f^1(x) = f(x) \leq f^0(x) = x$. Now a straightforward induction using monotonicity shows $f^{i+1}(x) \leq f^i(x)$ for all i so that the $f^i(x)$ form a descending chain. Since M is noetherian, the chain of the $f^i(x)$ has to become stationary, i.e., there is some k such that $f^k(x) = f^{k+1}(x)$. But then $f^j(x) = f^k(x)$ for all $j \geq k$ and hence $f^k(x) = \text{glb}\ X$, where $X \stackrel{\text{def}}{=} \{f^i(x) : i \in \mathbb{N}\}$, so that $x_\infty = f^k(x)$. But then $x_\infty = f^k(x) = f^{k+1}(x) = f(f^k(x)) = f(x_\infty)$, i.e., x_∞ is a fixpoint of f.
(2) A straightforward induction using monotonicity of f shows that $x_\infty \leq f^i(y) \leq f^i(x)$ for all $i \in \mathbb{N}$. Hence x_∞ is a lower bound for $Y \stackrel{\text{def}}{=} \{f^i(y) : i \in \mathbb{N}\}$. Let z be another lower bound for Y. Then z is also a lower bound for X defined in (1) and hence $z \leq x_\infty$. Hence $x_\infty = \text{glb}\ Y = y_\infty$.
(3) Trivially, $f(\top) \leq \top$, and hence $\top_\infty$ exists by (1). Let x be a fixpoint of f. A straightforward induction using monotonicity of f shows that $x \leq f^i(\top)$ for all $i \in \mathbb{N}$, so that x is a lower bound for $\{f^i(\top) : i \in \mathbb{N}\}$. But then $x \leq \top_\infty = \text{glb}\ \{f^i(\top) : i \in \mathbb{N}\}$. □

A similar theorem has been stated by Cai, Paige 1989.

Corollary 5. *If $\top_\infty \leq x$ for some $x \in M$ then $\top_\infty = x_\infty$.*

Proof. By (2) of the above theorem. □

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

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

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:

$$inf(y) = (x_\infty \leq y \leq x) \bullet \mathsf{if}\ y = f(y)\ \mathsf{then}\ y\ \mathsf{else}\ inf(f(y))\ \mathsf{fi}\ .$$

This recursion terminates for every y satisfying $f(y) \leq y$, since monotonicity then also shows $f(f(y)) \leq f(y)$, 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 $(\mathcal{P}(A), \subseteq)$ is noetherian with greatest element A. Therefore A_∞ exists. Moreover, we have

Corollary 6. $A_{|A|} = A_\infty$.

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

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 $W \subseteq A$)

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

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

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

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

$$hascycle \;=\; hcy(A)\ , \tag{76}$$

where

$$\begin{aligned} hcy(W) \;=\; (A_\infty \subseteq W \wedge W\,;R \subseteq W) \bullet \\ \mathsf{if}\ src(W) = \emptyset\ \mathsf{then}\ W \neq \emptyset\ \mathsf{else}\ hcy(W\backslash src(W))\ \mathsf{fi}\ , \end{aligned} \tag{77}$$

which is one of the classical algorithms which works by successive removal of sources (see e.g. Berghammer 1986). Note that Lemma 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.

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

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

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

$$x\,;in(W) \stackrel{\text{def}}{=} |R\,;x \cap W| \ . \tag{78}$$

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

$$src(W) = W \cap in(W);0 \ . \tag{79}$$

In a 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

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

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

$$x\,;(f \wr g) \stackrel{\text{def}}{=} (x\,;f) \wr (x\,;g) \ . \tag{80}$$

Then

$$in(W\backslash src(W)) = in(W) - in(src(W)) \ . \tag{81}$$

For the computation of in we observe that

$$in(\emptyset) = \mathbf{0} \ , \tag{82}$$

where

$$x\,;\mathbf{0} \stackrel{\text{def}}{=} 0 \ . \tag{83}$$

If $S \neq \emptyset$ and $q \in S$ is arbitrary we have

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

where

$$x\,;\,in(q) \;=\; \mathsf{if}\ q\,;\,R\,;\,x\ \mathsf{then}\ 1\ \mathsf{else}\ 0\ \mathsf{fi}\ . \tag{84}$$

Then

$$in(S) = in(q) + in(S\backslash q)\ . \tag{85}$$

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. Partsch 1990). We set, for $S \subseteq W \subseteq A$ and relation f,

$$hc(W,S,f) \stackrel{\text{def}}{=} (S = src(W) \wedge f = in(W)) \bullet hcy(W)\ , \tag{86}$$

with the embedding

$$hcy(W) \;=\; hc(W, src(W), in(W))\ . \tag{87}$$

Now

$$
\begin{array}{ll}
& hc(W,S,f) \\
= & \quad \{\![\ \text{definitions}\ ]\!\} \\
& \mathsf{if}\ src(W) = \emptyset\ \mathsf{then}\ W \neq \emptyset\ \mathsf{else}\ hcy(W\backslash src(W)) \\
= & \quad \{\![\ \text{assertion}\ ]\!\} \\
& \mathsf{if}\ S = \emptyset\ \mathsf{then}\ W \neq \emptyset\ \mathsf{else}\ hcy(W\backslash S) \\
= & \quad \{\![\ \text{embedding}\ ]\!\} \\
& \mathsf{if}\ S = \emptyset\ \mathsf{then}\ W \neq \emptyset\ \mathsf{else}\ hc(W\backslash S\ ,\ src(W\backslash S)\ ,\ in(W\backslash S)) \\
= & \quad \{\![\ \text{introducing auxiliaries}\ ]\!\} \\
& \mathsf{if}\ S = \emptyset\ \mathsf{then}\ W \neq \emptyset \\
& \qquad \mathsf{else}\ \mathsf{let}\ T \stackrel{\text{def}}{=} W\backslash S \\
& \qquad\quad \mathsf{let}\ g \stackrel{\text{def}}{=} in(T) \\
& \qquad\quad \mathsf{in}\ \ hc(T, src(T), g)\ \ \mathsf{fi} \\
= & \quad \{\![\ \text{by (81) and (79)}\ ]\!\} \\
& \mathsf{if}\ S = \emptyset\ \mathsf{then}\ W \neq \emptyset \\
& \qquad \mathsf{else}\ \mathsf{let}\ T \stackrel{\text{def}}{=} W\backslash S \\
& \qquad\quad \mathsf{let}\ g \stackrel{\text{def}}{=} f - in(S) \\
& \qquad\quad \mathsf{in}\ \ hc(T,\ T \cap g\,;\,0,\ g)\ \ \mathsf{fi}\ .
\end{array}
$$

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. Partsch 1990).

4 Conclusion

The calculus of formal languages and relations has proved to speed up derivations, in particular the way from "non-operational" specifications involving the closures R^* and $R^{\Rightarrow}$ to first recursive solutions. But 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 should demonstrate the viability of the approach in more complicated examples are under way. Also, we are working on 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 the anonymous referees for a number of valuable remarks.

References

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, H. Wössner: The Munich project CIP. Volume I: The wide spectrum language CIP-L. Lecture Notes in Computer Science **183**. Berlin: Springer 1985

F.L. Bauer, B. Möller, H. Partsch, P. Pepper: Formal program construction by transformations — Computer-aided, Intuition-guided Programming. IEEE Transactions on Software Engineering **15**, 165–180 (1989)

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

R. Bird: Lectures on constructive functional programming. In M. Broy (ed.): Constructive methods in computing science. NATO ASI Series. Series F: Computer and systems sciences **55**. Berlin: Springer 1989, 151–216

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

J. Cai, R. Paige: Program derivation by fixed point computation. Science of Computer Programming **11**, 197–261 (1989)

C.J. Date: An introduction to database systems. Vol. I, 4th edition. Reading, Mass.: Addison-Wesley 1988

M. Gondran, M. Minoux: Graphes et algorithmes. Paris: Eyrolles 1979

P. Lescanne: Modèles non déterministes de types abstraits. R.A.I.R.O. Informatique théorique **16**, 225–244 (1982)

Z. Manna: Mathematical theory of computation. New York: McGraw-Hill 1974

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

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

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

H.A. Partsch: Specification and transformation of programs — A formal approach to software development. Berlin: Springer 1990

G. Schmidt, T. Ströhlein: Relationen und Graphen. Berlin: Springer 1989. English version: Relations and graphs (forthcoming)

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