Constructing Programs from Specifications
B. Möller (Ed.)
Elsevier Science Publishers B.V. (North-Holland)
1991 IFIP

Relations as a Program Development Language

Bernhard Möller

Institut für Mathematik, Universität Augsburg, Universitätsstr. 2, W-8900 Augsburg, Germany

Abstract
We use n-ary higher-order relations between nested tuples as elements of a language in which to specify and develop programs. The main emphasis is laid on algebraic laws for carrying out the transformations from a specification into a program. Fixpoint semantics of general recursion comes cheap without domain theory, since the set of all relations of a given arity forms a complete lattice under set inclusion. Relational union provides angelic non-deterministic choice. The Booleans can be replaced by the (only possible) two nullary relations. This leads to simple definitions of assertions, conditional, and guards. Besides relational composition we introduce a special case of relational join which corresponds to path concatenation in directed graphs. For the case of unary relations, i.e., sets, these operations give convenient definitions of image, inverse image, and restriction. Closure operations are defined as least fixpoints of certain recursions; consequently, we can prove properties about them by computational induction. These properties are used in the derivation of a simple reachability algorithm on directed graphs.

1 INTRODUCTION

The transformational or calculational approach to program development has by now a long tradition [8,2,3,21,6]. There one starts from a (possibly non-executable) specification and transforms it into a (hopefully efficient) program using semantics-preserving rules. While in the beginning only deterministic programs were considered, [2] provided a first non-deterministic language CIP-L. However, the rules for CIP-L suffered from the amount of side conditions, especially those concerning non-determinacy, attached to them. For that reason, search for a simpler semantic basis had been going on for some time.

Within CIP-L, a particular data type, viz. that of partial maps received special attention in connection with the derivation of pointer [25,4] and matrix [27] algorithms. The idea was to find a set of algebraic laws in the style of [21,6] to allow convenient manipulation of maps. However, the sets of laws presented in [25,4,27] are quite diverse and unsystematic. Moreover, map union, the central operation, is partial, since the union of two maps may yield a non-functional relation. The incentive for the present paper therefore was the search of a more uniform treatment of map algebra within the framework of relations which encompasses partial maps as a special case. However, it turned out that within this framework also the above-mentioned simpler basis for a program development language could be found.

Of course, the idea of working in a relational framework is by no means new [22,11,14,1]. Let us therefore point out the main particularities of our approach:

(1) We consider relations of arbitrary arities. Arities ≥ 2 correspond to non-deterministic functions with tuples as arguments and/or results. Relations of arity 1 represent types, i.e., sets of single elements. The only two nullary relations (the singleton relation consisting just of the empty tuple and the empty relation) play the role of the Boolean values. This also allows easy definitions of assertions, conditional, and guards.

(2) We allow nested tuples as elements of relations.

(3) Relations may be of higher order, i.e., contain other relations as tuple components. This allows also parameterized and dependent types.

Essential operations on relations are (besides union, intersection, and difference) junction and join. Junction encompasses concatenation and composition; as special cases 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.

All operations are montonic w.r.t. relation inclusion. Hence the Knaster-Tarski fixpoint theorem provides semantics for recursive definitions of relations. By (1) above we get recursion on types as well as on non-deterministic functions (where union represents angelic non-deterministic choice). (2) allows general tree-like data types to be defined in this way. The principle of computational induction provides proofs about recursively defined types and functions.

The approach is illustrated with the derivation of a simple reachability algorithm on directed graphs.

Most proofs are straightforward and therefore omitted.

2 OPERATIONS ON SETS

Given a set A we denote by $\mathcal{P}(A)$ its **powerset**. Frequently, we will extend set-valued operations

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

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

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

defined by

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

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:

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

To save braces, we identify a singleton set with its only element whenever no ambiguities arise. In particular, for $x \in V$, $S \in \mathcal{P}(V)$, and $? \in \{\cup, \cap, \setminus\}$ we abbreviate $\{x\}?S$ and $S?\{x\}$ by $x?S$ and $S?x$, resp.

3 TUPLES, LANGUAGES, AND RELATIONS

We are working with general, n-ary relations. Since relations are subsets of cartesian products, we need some notation for their elements. We denote tuples as sequences of components delimited by the brackets $\langle$ and $\rangle$, with $\langle\rangle$ standing for the empty tuple. Tuples may be nested, e.g.

$$\langle\langle\rangle\rangle\,, \qquad \langle\langle\rangle,\langle\rangle\rangle\,, \qquad \langle\langle\langle\rangle\rangle,\langle\langle\rangle,\langle\rangle\rangle\rangle$$

Tuple concatenation is defined by

$$\langle a_1, \ldots, a_m\rangle \bullet \langle b_1, \ldots, b_n\rangle \overset{\text{def}}{=} \langle a_1, \ldots, a_m, b_1, \ldots, b_n\rangle \quad (5)$$

It is associative, with $\langle\rangle$ as the neutral element:

$$\begin{aligned} u \bullet (v \bullet w) &= (u \bullet v) \bullet w & (6)\\ \langle\rangle \bullet u &= u & (7)\\ u \bullet \langle\rangle &= u & (8) \end{aligned}$$

The **reverse** of a tuple $u = \langle a_1, \ldots, a_n\rangle$ is the tuple $u^{-1} \overset{\text{def}}{=} \langle a_n, \ldots, a_1\rangle$. We have $u^{-1} = u$ for singleton tuples and $\langle\rangle^{-1} = \langle\rangle$.

A **(formal) language** is a set of tuples. Associativity of concatenation also holds for languages:

$$U \bullet (V \bullet W) = (U \bullet V) \bullet W \quad (9)$$

The language $\varepsilon \overset{\text{def}}{=} \{\langle\rangle\}$ is its neutral element:

$$\varepsilon \bullet U = U = U \bullet \varepsilon \quad (10)$$

We extend concatenation to finite families $(U_i)_{i\in[1:n]}$ of languages by setting for $j, k \in [1:n]$

$$\mathop{\bullet}_{i=j}^{k} U_i \overset{\text{def}}{=} \begin{cases} \varepsilon & \text{if } j > k \\ (\mathop{\bullet}\limits_{i=j}^{k-1} U_i) \bullet U_k & \text{otherwise} \end{cases} \quad (11)$$

For a language U we define the powers $U^{(n)}$ ($n \in \mathbb{N}$) by

$$U^{(n)} \overset{\text{def}}{=} \mathop{\bullet}_{i=1}^{n} U \quad (12)$$

In particular,

$$\begin{aligned} U^{(0)} &= \varepsilon & (13)\\ U^{(1)} &= U & (14) \end{aligned}$$

Then

$$U^{(*)} \overset{\text{def}}{=} \bigcup_{n\in\mathbb{N}} U^{(n)} \quad (15)$$

is the language of all tuples formed by concatenating finitely many elements of U. In formal language theory the parentheses around the superscripts usually are dropped; however, we are going to use the unparenthesized notation with a different meaning in defining the reflexive transitive closure of a binary relation.

The **diagonals** V^{Δ_n} ($n \in \mathbb{N}$) on a language V are defined by

$$V^{\Delta_n} \overset{\text{def}}{=} \bigcup_{x \in V} x^{(n)} \tag{16}$$

In particular,

$$V^{\Delta_0} = \begin{cases} \varepsilon & \text{if } V \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \tag{17}$$

$$V^{\Delta_1} = V \tag{18}$$

We have

$$(V^{\Delta_n})^{-1} = (V^{-1})^{\Delta_n} \tag{19}$$

Consider now some set U from which the tuple components are taken. The **length** of a tuple u is denoted by $|u|$ and defined by

$$|\varepsilon| = 0 \tag{20}$$

$$|\langle a \rangle| = 1 \qquad (a \in U) \tag{21}$$

$$|u \bullet v| = |u| + |v| \tag{22}$$

The singleton tuple consisting just of the i-th ($1 \leq i \leq |u|$) component of a tuple u is denoted by u_i. So

$$u = \overset{|u|}{\underset{i=1}{\bullet}} u_i \tag{23}$$

A **relation** R is a language such that all tuples in R have equal length. If $R \neq \emptyset$ this length is called the **arity** of R and is denoted by $\operatorname{ar} R$. The empty relation $\emptyset$ is considered to have any arity required from the context. With this convention, there are only two 0-ary relations, viz. $\emptyset$ and ε. Note also that, for relations $R, S \neq \emptyset$ the language $R \bullet S$ is again a relation with

$$\operatorname{ar}(R \bullet S) = \operatorname{ar} R + \operatorname{ar} S \tag{24}$$

Given a relation R, the language R^{-1} is called the **converse** relation of R. For unary R we have $R^{-1} = R$. Moreover, $\varepsilon^{-1} = \varepsilon$.

4 A HIERARCHICAL UNIVERSE OF DISCOURSE

We now want to construct a hierarchy of sets from which our relations will be taken. The levels in the hierarchy correspond to those in typed λ-calculus: Objects of level n will roughly be $\underbrace{\text{languages of} \ldots \text{of languages}}_{n}$. More precisely, the elements of a language of level $n+1$ are nested tuples of languages from level n and below. In the special case of relations this leads to relations of higher order. First we give a construction for the set $V^{\diamond}$ of nested tuples over some set V: We define

$$V^{\diamond} = \bigcup_{i \in \mathbb{N}} V_i \tag{25}$$

where

$$V_0 = V \tag{26}$$

$$V_{i+1} = \langle \bigcup_{k=0}^{i} V_k \rangle^{(*)} \tag{27}$$

So V_{i+1} contains all tuples of nesting depth $i+1$, and hence $V^\diamond$ consists of all (finitely) nested tuples.

Now we can construct our hierarchy of universes $\mathcal{U}_i (i \in \mathbb{N})$. At the lowest level we start with a nonempty denumerable set $\mathcal{U}$ of **urelements** which are considered atomic. Then we define

$$\mathcal{U}_0 = \mathcal{U} \tag{28}$$

$$\mathcal{U}_{i+1} = \mathcal{P}((\bigcup_{k=0}^{i} \mathcal{U}_k)^\diamond) \tag{29}$$

According to this definition, $\mathcal{U}_{i+1}$ consists of all languages the elements of which are nested tuples of elements of any $\mathcal{U}_k$ with $k \leq i$. In the sequel, we are only considering languages and relations from one of these universes.

5 JUNCTION AND COMPOSITION

As auxiliary operation on tuples we need the **detachment** $x \rfloor y$ of a tuple x from the left of another tuple y. To make the algebra nicer, we let $\rfloor$ return a *set* of tuples of cardinality at most 1. If x is a prefix of y then $x \rfloor y$ contains the remainder of y after x. Otherwise, if y is a prefix of x the operation returns ε, the result of erasing x from y "as far as possible". In all other cases it returns the "error value" $\emptyset$. We define it inductively by

$$x \rfloor \langle\rangle = \varepsilon \tag{30}$$

$$\langle\rangle \rfloor y = y \tag{31}$$

$$(u \bullet x) \rfloor (v \bullet y) = \begin{cases} x \rfloor y & \text{if } u = v \\ \emptyset & \text{otherwise} \end{cases} \qquad (|u| = 1 = |v|) \tag{32}$$

A related operation on languages is forming the left factor as known in formal language theory. Analogously, we define the **detachment** $x \lfloor y$ of a tuple y from the right of a tuple x:

$$x \lfloor y = (y^{-1} \rfloor x^{-1})^{-1} \tag{33}$$

An important dyadic operation on tuples is the junction along common elements. It is a generalization of the operation that takes place when two singleton binary relations are composed in the usual sense: Denoting relational composition by ; we have

$$\{\langle a, b\rangle\} ; \{\langle c, d\rangle\} = \begin{cases} \{\langle a, d\rangle\} & \text{if } b = c \\ \emptyset & \text{otherwise} \end{cases}$$

This means that a "common factor" of length 1 is erased from the two tuples and the remainders are glued together to make up the result. If there is no common factor, the tuples are not composable and the empty relation results. We want to generalize this to allow composition along k common elements; then e.g. a relation with two results may be composed with another one having two arguments. We call the respective operation **junction** and define it using detachment:

$$x \,ⓚ\, y \stackrel{\text{def}}{=} \bigcup_{|z|=k} (x \lfloor z) \bullet (z \rfloor y) \tag{34}$$

So this operation erases a common part of length k at the right of x and at the left of y and concatenates the remainders. If no common part of length k exists, again the "error value" $\emptyset$ results. From the definition the following properties are immediate:

$$x \,\textcircled{0}\, y = \{x \bullet y\} \tag{35}$$

$$x \,\textcircled{k}\, \varepsilon = \mathop{\bullet}_{i=1}^{|x|-k} x_i \tag{36}$$

$$\varepsilon \,\textcircled{k}\, x = \mathop{\bullet}_{i=k+1}^{|x|} x_i \tag{37}$$

$$\varepsilon \,\textcircled{k}\, \varepsilon = \varepsilon \tag{38}$$

So a k-junction with ε erases up to k components from the respective side of a tuple.

For $|x| \le k \le |y|$ we have

$$x \,\textcircled{k}\, y = \begin{cases} u & \text{if } \exists\, v : |v \bullet x| = k \wedge v \bullet x \bullet u = y \\ \emptyset & \text{otherwise} \end{cases} \tag{39}$$

Hence $\bigcup_{k=|x|}^{|y|} x \,\textcircled{k}\, y$ is the set of remainders of y after occurrences of x in y; this is useful in specifying pattern matching problems.

Junctions associate:

$$x \,\textcircled{k}\, (y \,\textcircled{n}\, z) = (x \,\textcircled{k}\, y) \,\textcircled{n}\, z \quad \Leftarrow \quad |y| \ge k + n \tag{40}$$

As a special case of junction on languages we obtain concatenation:

$$U \,\textcircled{0}\, V = U \bullet V \tag{41}$$

Junction, and hence also concatenation, is contravariant with reversal:

$$(U \,\textcircled{k}\, V)^{-1} = V^{-1} \,\textcircled{k}\, U^{-1} \tag{42}$$

For relation R with $\mathsf{ar}\, R > 0$ and $k \ge 0$ we define, with $n \overset{\text{def}}{=} max(\mathsf{ar}\, R - k, 0)$,

$$\mathsf{dom}_k\, R = R \,\textcircled{n}\, \varepsilon \tag{43}$$

$$\mathsf{cod}_k\, R = \varepsilon \,\textcircled{n}\, R \tag{44}$$

These are the **domain** and **codomain** of length k of R, i.e., the languages of all k first and k last components of tuples occurring in R. In particular

$$\mathsf{dom}_0\, R = \begin{cases} \varepsilon & \text{if } R \ne \emptyset \\ \emptyset & \text{otherwise} \end{cases} \tag{45}$$

$$\mathsf{dom}_k\, R = \mathsf{cod}_k\, R = R \quad \Leftarrow \quad k \ge \mathsf{ar}\, R \tag{46}$$

$$\mathsf{dom}_k\, \varepsilon = \mathsf{cod}_k\, \varepsilon = \varepsilon \tag{47}$$

As abbreviations we use

$$\mathsf{dom}\, R \overset{\text{def}}{=} \mathsf{dom}_1\, R \tag{48}$$

$$\mathsf{cod}\, R \overset{\text{def}}{=} \mathsf{cod}_1\, R \tag{49}$$

Suitable diagonals are neutral w.r.t. junction: Suppose R is a relation and $k \le \mathsf{ar}\, R$. Then

$$(\mathsf{dom}_k\, R)^{\Delta_2} \,\textcircled{k}\, R = R \tag{50}$$

$$R \,\textcircled{k}\, (\mathsf{cod}_k\, R)^{\Delta_2} = R \tag{51}$$

For diagonals, junction leads to a kind of intersection [31]: Suppose R, S are relations with $\mathsf{ar}\, R = \mathsf{ar}\, S = k$. Then

$$R^{\Delta_m} \,\textcircled{k}\, S^{\Delta_n} = (R \cap S)^{\Delta_{m+n-2}} \quad \Leftarrow \quad m, n > 0 \tag{52}$$

Junction provides a flexible way of composing relations. Consider e.g. the following relations:

$$\begin{aligned} R &\stackrel{\text{def}}{=} \{\langle a,b,a+b,a-b\rangle \mid a,b \in \mathbb{N}\} \\ S &\stackrel{\text{def}}{=} \{\langle a,b,a*b\rangle \mid a,b \in \mathbb{N}\} \\ T &\stackrel{\text{def}}{=} \{\langle a,a^2\rangle \mid a \in \mathbb{N}\} \end{aligned}$$

Then

$$R \circled{2} S = \{\langle a,b,(a+b)*(a-b)\rangle \mid a,b \in \mathbb{N}\}$$

whereas

$$S \circled{1} T = \{\langle a,b,(a*b)^2\rangle \mid a,b \in \mathbb{N}\}$$

On binary relations $\circled{1}$ coincides with usual relational composition (see e.g. [29]. For that reason, we introduce the abbreviation

$$U\,;V \stackrel{\text{def}}{=} U \circled{1} V \tag{53}$$

and term it **composition** in the sequel. Again we obtain a distributive, monotonic and strict operation.

Junctions associate:

$$U \circled{k} (V \circled{n} W) = (U \circled{k} V) \circled{n} W \;\Leftarrow\; \forall\, y \in V : |y| \geq k+n \tag{54}$$

The following association properties are derived from the general law above:

$$\begin{aligned} U\,;(V\,;W) &= (U\,;V)\,;W &\Leftarrow\; \forall\, y \in V : |y| \geq 2 \qquad (55) \\ U \bullet (V\,;W) &= (U \bullet V)\,;W &\Leftarrow\; \forall\, y \in V : |y| \geq 1 \qquad (56) \\ U\,;(V \bullet W) &= (U\,;V) \bullet W &\Leftarrow\; \forall\, y \in V : |y| \geq 1 \qquad (57) \end{aligned}$$

We shall omit parentheses whenever one of these laws applies.

Diagonals enjoy further associativities [31]: Let R, S be relations with $\mathsf{ar}\,R = \mathsf{ar}\,S = k$. Then

$$R^{\Delta_m} \circled{k} (S \circled{k} R^{\Delta_n}) = (R^{\Delta_m} \circled{k} S) \circled{k} R^{\Delta_n} \;\Leftarrow\; m = n = 1 \vee (m > 1 \wedge n > 1) \tag{58}$$

Proof: For $m = n = 1$ the result is immediate from (52) and (17). Assume now $m > 1$ and $n > 1$.

$$\begin{aligned} & R^{\Delta_m} \circled{k} (S \circled{k} R^{\Delta_n}) \\ = \quad & \{\!\!\{ \text{ by } (18{,}52) \}\!\!\} \\ & R^{\Delta_m} \circled{k} (S \cap R)^{\Delta_{n-1}} \\ = \quad & \{\!\!\{ \text{ by } (52) \}\!\!\} \\ & (R \cap S)^{\Delta_{m+n-3}} \\ = \quad & \{\!\!\{ \text{ by } (52) \}\!\!\} \\ & (R \cap S)^{\Delta_{m-1}} \circled{k} R^{\Delta_n} \\ = \quad & \{\!\!\{ \text{ by } (18{,}52) \}\!\!\} \\ & (R^{\Delta_m} \circled{k} S) \circled{k} R^{\Delta_n} \end{aligned}$$

■

Further interesting special cases of junction arise when the junction is formed w.r.t. the minimum of the arities involved. Suppose $k = \mathsf{ar}\,R \leq \mathsf{ar}\,S$. Then

$$R \circled{k} S = \bigcup_{x \in R} \{v : x \bullet v \in S\}$$

In other words, $R \textcircled{k} S$ is the image of R under S. Likewise, if $k = \mathsf{ar}\, T \leq \mathsf{ar}\, S$, then $S \textcircled{k} T$ is the inverse image of T under S.

Suppose now $\mathsf{ar}\, R = k = \mathsf{ar}\, S$ and $|x| = k = |y|$. Then

$$R \textcircled{k} S = \begin{cases} \varepsilon & \text{if } R \cap S \neq \emptyset \\ \emptyset & \text{if } R \cap S = \emptyset \end{cases} \tag{59}$$

$$R \textcircled{k} R = \begin{cases} \varepsilon & \text{if } R \neq \emptyset \\ \emptyset & \text{if } R = \emptyset \end{cases} \tag{60}$$

$$x \textcircled{k} R = R \textcircled{k} x = \begin{cases} \varepsilon & \text{if } x \in R \\ \emptyset & \text{if } x \notin R \end{cases} \tag{61}$$

$$x \textcircled{k} y = y \textcircled{k} x = \begin{cases} \varepsilon & \text{if } x = y \\ \emptyset & \text{if } x \neq y \end{cases} \tag{62}$$

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

$$R \neq \emptyset = R \textcircled{k} R \tag{63}$$

$$x \in R = x \textcircled{k} R \tag{64}$$

$$x = y = x \textcircled{k} y \tag{65}$$

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

$$x \,;\, R \,;\, y = \begin{cases} \varepsilon & \text{if } x \bullet y \in R \\ \emptyset & \text{otherwise} \end{cases} \tag{66}$$

6 CIRCULAR SHIFT

The composition via junction alone still is not flexible enough, since it does not allow the consideration of "inner" tuple elements. Consider again

$$R \overset{\text{def}}{=} \{\langle a, b, a+b, a-b \rangle \mid a, b \in \mathbb{N}\}$$
$$S \overset{\text{def}}{=} \{\langle a, b, a*b \rangle \mid a, b \in \mathbb{N}\}$$

Then it is not possible to feed the "results" $a * b$ of S into the second "argument" b of R using the operations we have so far. To remedy this we introduce the circular left and right shifts $\hookleftarrow$ and $\hookrightarrow$ for rotating the components of a tuple. They are defined by

$$\hookleftarrow x = \varepsilon \textcircled{1} x \bullet x_1 \tag{67}$$

$$\hookrightarrow x = x_{|x|} \bullet x \textcircled{1} \varepsilon \tag{68}$$

Using these, our task is now solved by the expression

$$\hookrightarrow (S \textcircled{1} \hookleftarrow R)$$

Generally, the junction of P and Q along the k inner components ocurring immediately after position n within the tuples of Q is formed by

$$\hookrightarrow^n (P \textcircled{k} \hookleftarrow^n Q)$$

7 EXTENSIONALITY

We may decompose relations in a domain-oriented and a range-oriented way: For $k \leq \mathsf{ar}\, R$,

$$R = \bigcup_{x \in \mathsf{dom}_k R} x \bullet x \circledk R \qquad \text{(domain-oriented)} \tag{69}$$

$$R = \bigcup_{x \in \mathsf{cod}_k R} R \circledk x \bullet x \qquad \text{(range-oriented)} \tag{70}$$

If we take as our relation R the partial map representing the dereferencing operation on a von Neumann storage, the range-oriented decomposition of R plays a prominent role in the derivation of the garbage collection algorithm presented in [4]; it leads to a chaining of all cells which point to a given other cell, so that all these pointers can be adjusted in one pass in the compactifying relocation.

From the decomposition properties we also immediately get extensionality and co-extensionality:

$$(\forall\, x \in \mathsf{dom}_k R : x \circledk R = x \circledk S) \;\Rightarrow\; R = S \tag{71}$$

$$(\forall\, x \in \mathsf{cod}_k R : R \circledk x = S \circledk x) \;\Rightarrow\; R = S \tag{72}$$

These properties will be used in "pointwise" calculation of the behaviour of relations.

8 ABSTRACTIONS

We already have used unions of families of languages. This mechanism also provides us with a relational analogue of λ-abstractions: The typed abstraction

$$\lambda\, a \in \mathbb{N} \,.\, a^2 \tag{73}$$

can be expressed as

$$\bigcup_{a \in \mathbb{N}} \langle a, a^2 \rangle \tag{74}$$

Application is then obtained using composition:

$$\begin{array}{ll} & \langle 5 \rangle \,;\, \bigcup_{a \in \mathbb{N}} \langle a, a^2 \rangle \\ = & \quad \{\!\![\text{ distributivity }]\!\!\} \\ & \bigcup_{a \in \mathbb{N}} \langle 5 \rangle \,;\, \langle a \rangle \bullet \langle a^2 \rangle \\ = & \quad \{\!\![\text{ since } \langle 5 \rangle \,;\, \langle a \rangle = \emptyset \text{ for } a \neq 5 \,]\!\!\} \\ & \langle 5 \rangle \,;\, \langle 5 \rangle \bullet \langle 5^2 \rangle \\ = & \varepsilon \bullet \langle 25 \rangle \\ = & \langle 25 \rangle \end{array}$$

Generally we define for a language V and relational expression R in which the identifier x may occur

$$\lambda\, x \in V \,.\, R \;\overset{\text{def}}{=}\; \bigcup_{x \in V} x \bullet R \tag{75}$$

An example of a higher-order abstraction is the cartesian product former

$$\lambda\, V \in \mathcal{U}_i \,.\, \lambda\, W \in \mathcal{U}_i \,.\, V \bullet W \qquad (i > 0)$$

9 PARAMETERIZED TYPES AND POLYMORPHISM

So far we have used only "simple" parameters in abstractions. By the rich structure of our universe, however, we may also use sets, i.e., types, as parameters. In this way we obtain both parameterized types, such as the type of sequences,

$$\lambda\, A \in \mathcal{U}_i \,.\, A^{(*)} \qquad (i > 0)$$

and polymorphic relations, such as a doubling function

$$\lambda\, A \in \mathcal{U}_i \,.\, \lambda\, x \in A \,.\, x \bullet x \qquad (i > 0)$$

More intricate examples are obtained in connection with recursion (see below).

Note that this form of polymorphism is not fully general [23], since the abstraction requires type information. Nevertheless there are many useful applications without any additional theory.

10 PARTIALITY AND STRICTNESS

The above definition of abstractions might give the impression that only left-total relations can be defined that way. However, the body R in

$$\lambda\, x \in V \,.\, R$$

may for certain values of x evaluate to the "error element" $\emptyset$, i.e., to no proper value at all. Hence partial relations with a domain smaller than V may result.

In composing such partial relations, "undefinedness" propagates, as is immediate from the definition of junction. Hence composition is strict, i.e., for functional relations a call-by-value semantics results.

11 NON-DETERMINACY

Since relations may be one-to-many, they immediately provide a means for handling non-determinacy. Union plays the role of non-deterministic choice. In fact, this choice is "angelic" in that the union of two relations provides image values for an argument value whenever one of them does. This is mirrored by the neutrality of $\emptyset$ w.r.t. union, since $\emptyset$ plays the role of $\perp$ as known from denotational semantics.

As pointed out in [7], some caution is necessary with non-deterministic choice at the level of functions. Let us briefly explain the reason for the paradox derived in that paper. The choice operator is denoted by $[]$ there, and postulates equivalent to the following ones are put forward (in functional notation):

$$\begin{array}{ll}(\mathrm{A}) & f(x \,[]\, y) \;=\; f(x) \,[]\, f(y) \\ (\mathrm{B}) & (f \,[]\, g)(x) \;=\; f(x) \,[]\, g(x)\end{array}$$

(A) applies also to higher-order functions f. Moreover, the principle of extensionality is required to hold. Together with (B) this implies that

$$(\mathrm{C})\ f \,[]\, g \;=\; h \quad \text{where } h \overset{\text{def}}{=} \lambda\, x \,.\, f(x) \,[]\, g(x)$$

Consider now the functions $id, not : \mathbb{B} \to \mathbb{B}$, where $\mathbb{B} = \{\mathsf{O}, \mathsf{L}\}$, and $tab : (\mathbb{B} \to \mathbb{B}) \to \langle \mathbb{B} \rangle \bullet \langle \mathbb{B} \rangle$ defined by

$$tab(f) \overset{\text{def}}{=} \langle f(\mathsf{O}), f(\mathsf{L}) \rangle$$

Now we have

$$
\begin{array}{cl}
 & \langle \mathsf{O},\mathsf{L}\rangle \mathbin{[]} \langle \mathsf{L},\mathsf{O}\rangle \\
= & \quad \{\!\![\text{ definition of } \mathit{tab} \ ]\!\!\} \\
 & \mathit{tab}(\mathit{id}) \mathbin{[]} \mathit{tab}(\mathit{not}) \\
= & \quad \{\!\![\text{ by (B) }]\!\!\} \\
 & \mathit{tab}(\mathit{id} \mathbin{[]} \mathit{not}) \\
= & \quad \{\!\![\text{ by (C) }]\!\!\} \\
 & \mathit{tab}(h) \\
= & \quad \{\!\![\text{ definition of } \mathit{tab} \ ]\!\!\} \\
 & \langle h(\mathsf{O}), h(\mathsf{L})\rangle \\
= & \quad \{\!\![\text{ definition of } h \ ]\!\!\} \\
 & (\mathit{id}(\mathsf{O}) \mathbin{[]} \mathit{not}(\mathsf{O})) \bullet (\mathit{id}(\mathsf{L}) \mathbin{[]} \mathit{not}(\mathsf{L})) \\
= & \quad \{\!\![\text{ definition of } \mathit{id}, \mathit{not} \ ]\!\!\} \\
 & \langle \mathsf{O} \mathbin{[]} \mathsf{L}\rangle \bullet \langle \mathsf{L} \mathbin{[]} \mathsf{O}\rangle \\
= & \quad \{\!\![\text{ by (A) applied to } \langle . \rangle \bullet \langle . \rangle \ ]\!\!\} \\
 & \langle \mathsf{O},\mathsf{L}\rangle \mathbin{[]} \langle \mathsf{O},\mathsf{O}\rangle \mathbin{[]} \langle \mathsf{L},\mathsf{L}\rangle \mathbin{[]} \langle \mathsf{O},\mathsf{O}\rangle
\end{array}
$$

This is an apparent paradox. It shows that postulating both (A) and (B) is inconsistent with extensionality. To retain extensionality, either (A) or (B) has to be dropped.

The relational framework cleanly distinguishes between

$$\{f\} \cup \{g\} = \{f,g\},$$

which should be the interpretation of a non-deterministic choice between f and g considered as objects, and

$$f \cup g = h$$

with h defined as in (C). If we evaluate the relational analogue of the expression

$$(\{f\} \cup \{g\})(x)$$

we obtain

$$
\begin{array}{cl}
 & \langle x\rangle \,;(\{f\} \cup \{g\}) \\
= & \langle x\rangle \,;\{f,g\} \\
= & \quad \{\!\![\text{ by (61) }]\!\!\} \\
 & \emptyset
\end{array}
$$

so that (B) does not hold. However, (A) is valid; relationally it reads

$$(\langle x\rangle \cup \langle y\rangle) \,; f = \langle x\rangle \,; f \;\cup\; \langle y\rangle \,; f \,,$$

which is just a special case of the distributivity of composition over union. Note that (B) entails that the parameter passing mechanism is call-time-choice [16,15,2].

12 ASSERTIONS AND QUANTIFIERS

As we have already seen in Section 5, 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 relational expressions yielding nullary relations **assertions**. Note that in this view "false" and "undefined" both are represented by $\emptyset$. Negation is defined by

$$\overline{\emptyset} \overset{\text{def}}{=} \varepsilon \tag{76}$$

$$\overline{\varepsilon} \overset{\text{def}}{=} \emptyset \tag{77}$$

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

$$B \bullet C = B \cap C \tag{78}$$

$$B \bullet B = B \tag{79}$$

$$B \bullet \overline{B} = \emptyset \tag{80}$$

$$B \cup \overline{B} = \varepsilon \tag{81}$$

$$\overline{B \bullet C} = \overline{B} \cup \overline{C} \tag{82}$$

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 relation R we have

$$B \bullet R = R \bullet B = \begin{cases} R & \text{if } B = \varepsilon \\ \emptyset & \text{if } B = \emptyset \end{cases} \tag{83}$$

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

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

in [24]. In fact, many of the properties of assertions listed there turn out to be special cases of properties of concatenation and composition. E.g., shifting of assertions through compositions can be done using

$$R \bullet B\,; S = R\,; B \bullet S \tag{84}$$

which follows from the above property.

Since there are only two nullary relations, in particular there are no "non-determinate" assertions. This avoids many of the side-conditions about determinacy that were necessary in CIP-L [2].

Using the concept of assertions we can also define universal and existential quantification as well as set comprehension and general non-deterministic choice. Assume that U is a language and B is a relational expression with $\mathsf{ar}\, B = 0$ which possibly involves the identifier x. Then we set

$$\exists\, x \in U : B \overset{\text{def}}{=} \bigcup_{x \in U} B \tag{85}$$

$$\forall\, x \in U : B \overset{\text{def}}{=} \bigcap_{x \in U} B \tag{86}$$

$$\{x \in U : B\} \overset{\text{def}}{=} \bigcup_{x \in U} B \bullet x \tag{87}$$

Since, as mentioned above, non-deterministic choice is represented by union in the relational framework, no separate concept is necessary for this. If desired, one could set

$$\mathsf{some}\ x \in U : B \overset{\text{def}}{=} \bigcup_{x \in U} B \bullet x \tag{88}$$

as well.

From these definitions, it is immediate that $\exists$, some , and $\{.\}$ distribute through union and hence are monotonic w.r.t. $\subseteq$ and strict w.r.t. $\emptyset$. However, $\forall$ does *not* distribute through union (and hence is not continuous); however, it is still monotonic.

13 DEPENDENT TYPES

Using comprehension, we can also define dependent types as used in Martin-Löf's theory [19]: Assume that A is a language and $B[x]$ is a relational expression possibly depending on the identifier x. We set

$$\prod_{x \in A} B[x] \overset{\text{def}}{=} \{f \subseteq \lambda x \in A \,.\, B[x] : f \,;\, f^{-1} \subseteq A^{\Delta_2} \wedge (\bigcup_{x \in A} B[x])^{\Delta_2} \subseteq f^{-1} \,;\, f\} \tag{89}$$

The assertion $f \subseteq \lambda x \in A \,.\, B[x]$ states that the value of x under f is an element of the appropriate set $B[x]$, while $f \,;\, f^{-1} \subseteq A^{\Delta_2} \wedge (\bigcup_{x \in A} B[x])^{\Delta_2} \subseteq f^{-1} \,;\, f$ requires f to be functional and left-total on A (see e.g.[28]).

14 CONDITIONAL AND GUARDS

Using assertions we can also define a conditional by

$$B \rightarrow \left\{ \begin{array}{c} R \\ S \end{array} \right\} \overset{\text{def}}{=} B \bullet R \cup \overline{B} \bullet S \tag{90}$$

for assertion B and relations R, S of equal types. This plays the role of the expression

$$\text{if } B \text{ then } R \text{ else } S \text{ fi} \,.$$

From this definition, the usual properties of the conditional (as stated e.g. in [20]) are easily proved using the properties of concatenation and composition. As an example we give an introduction/elimination rule for the conditional:

$$R = B \rightarrow \left\{ \begin{array}{c} R \\ R \end{array} \right\} \tag{91}$$

Note that no condition about the "definedness" of B is necessary here.

The concept of the conditional can be generalized to that of a guarded expression [2] analogous to Dijkstra's guarded commands [12]. Given assertions B_i and relations R_i $(1 \leq i \leq n)$ we define

$$\mathop{[\!]}_{i=1}^{n} (B_i \rightarrow R_i) \overset{\text{def}}{=} \bigcup_{i=1}^{n} B_i \bullet R_i \tag{92}$$

A similar definition appears in [21]. Note that, according to the general angelic view taken by the relational framework, the semantics differs substantially from Dijkstra's: If two guards open, but one of the corresponding branches is undefined, then the other branch is taken. The same angelic behaviour is shown w.r.t. evaluation of the guards in case one of the guards is undefined. The guarded expression obeys laws analogous to those for the conditional and additional ones concerning the relation between the branches. As a simple example we mention

$$(B \rightarrow R) \mathop{[\!]} (B \rightarrow S) \equiv B \rightarrow (R \cup S)$$

15 RECURSION

By definition, all sets of the shape $\mathcal{P}(V)$, where V is an arbitrary language, form complete lattices w.r.t. inclusion. Moreover, all operations except forming the relative complement are monotonic (most of them even continuous) w.r.t. inclusion. Hence standard fixpoint theory [18,30] applies to this setting. Given an identifier X for elements of $\mathcal{P}(V)$ and a context $C[X]$ which again yields elements of $\mathcal{P}(V)$ and which is monotonic in X, we denote by

$$\mu X \in \mathcal{P}(V) \,.\, C[X]$$

and

$$\nu X \in \mathcal{P}(V) \,.\, C[X]$$

its least and **greatest fixpoint**, resp. The type information "$\in \mathcal{P}(V)$" will frequently be omitted if it can be reconstructed from $C[X]$. The extension to simultaneous fixpoints is straightforward.

If $C[X]$ is continuous, i.e., distributes over unions of chains, we have the well-known iteration [17] for the least fixpoint:

$$\mu X \in \mathcal{P}(V) \,.\, C[X] \;=\; \bigcup_{i \in \mathbb{N}} C^i[\emptyset] \tag{93}$$

with

$$C^0[\emptyset] \;\stackrel{\text{def}}{=}\; \emptyset \tag{94}$$

$$C^{i+1}[\emptyset] \;\stackrel{\text{def}}{=}\; C[C^i[\emptyset]] \tag{95}$$

By dualizing the powerset lattice one obtains a corresponding iteration for the greatest fixpoint if $C[X]$ is co-continuous, i.e., distributes over intersections of chains.

The recursion mechanism also allows the formation of recursive data types. This is illustrated with the following examples.

Given a language $A \subseteq V^{(*)} \in \mathcal{U}_k$ $(k > 0)$ we may form

$$\mu X \in \mathcal{P}(V^{(*)}) \,.\; \varepsilon \,\cup\, A \bullet X$$

to obtain the language $A^{(*)}$. In particular, if A is a singleton set consisting of a singleton tuple (i.e. a letter), we obtain the language $\mathbb{N}$ of natural numbers. Since we have concatenation and union available, we obtain the full power of context-free languages (see [13]). By using a different recursion with tuple nesting, viz.

$$\mu X \in \mathcal{U}_k \,.\; \varepsilon \,\cup\, \langle A \bullet X \rangle$$

we obtain the type of finite lists with elements taken from A. Likewise, binary trees with node markings from A are given by

$$\mu X \in \mathcal{U}_k \,.\; \varepsilon \,\cup\, \langle X \bullet A \bullet X \rangle$$

Hence we also have the power of a language with variant records. Moreover, by making A into a parameter we get a recursive definition of a parametrized type:

$$\mu Y \,.\, \lambda A \in \mathcal{U}_k \,.\; \varepsilon \,\cup\, \langle (A\,;Y) \bullet A \bullet (A\,;Y) \rangle$$

A related definition is

$$\lambda A \in \mathcal{U}_k \,.\, \mu X \,.\; \varepsilon \,\cup\, \langle X \bullet A \bullet X \rangle$$

At this point we remark that our way of treating recursive types is quite similar to work reported in [1] and [26]. The main difference to [1] is that we use arbitrary, not just binary, relations; moreover, tupling is built-in in our system whereas it needs some coding in [1]. The main difference to [26] is that sorts and functions are kept separate there whereas they are treated symmetrically in our approach.

The language of repetition-free sequences over a set A of urelements is given by

$$\mu\, rf \in \mathcal{P}(\mathcal{U}_0) \,.\, \lambda A \in \mathcal{P}(\mathcal{U}_0) \,.\, \varepsilon \cup \bigcup_{x\in A} \langle x \rangle \bullet ((A\backslash x)\,;\, rf)$$

As an example of a recursive definition of a binary relation we take McCarthy's [20] non-deterministic function that returns for a natural number n any natural number k with $k \leq n$:

$$\mu\, le \,.\, \lambda x \in \mathbb{N} \,.\, x = 0 \rightarrow \left\{ \begin{array}{l} x \\ x \,\cup\, x\,;\, pred\,;\, le \end{array} \right\}$$

where *pred* is the predecessor relation on $\mathbb{N}$. In conventional notation this would read (with $[\!]$, i.e., angelic choice, instead of $\cup$):

$$\mu\, le \,.\, \lambda x \in \mathbb{N} \,.\, \mathsf{if}\ x = 0\ \mathsf{then}\ x\ \mathsf{else}\ x \,[\!]\, le(pred(x))\ \mathsf{fi}$$

Let us finally give an example of a higher-order recursion. We choose a polymorphic function for the iteration of a binary relation:

$$\lambda V \in \mathcal{U}_k \,.\, \mu\, it \,.\, \lambda x \in \mathbb{N} \,.\, \lambda R \in \mathcal{P}(V \bullet V) \,.\, x = 0 \rightarrow \left\{ \begin{array}{l} V^{\Delta_2} \\ R\,;\,(x\,;\, pred\,;\, it) \end{array} \right\}$$

Since we are applying standard fixpoint theory, we can also use the **principle of computational induction**:

Lemma 15.1
Let $C[X]$ be a monotonic context and $P[X]$ be a continuous predicate. Then from

$$\begin{array}{ll} P[\emptyset] & \text{and} \\ \forall X \,.\, P[X] \Rightarrow & P[C[X]] \end{array}$$

we may infer $P[\mu X \,.\, C[X]]$.

Here, a predicate P is **continuous** if for any chain

$$R_0 \subseteq R_1 \subseteq \cdots$$

of languages we have

$$(\forall i \in \mathbb{N} : P[R_i]) \Rightarrow P[\bigcup_{i\in\mathbb{N}} R_i]$$

For instance, $=$ and $\subseteq$ are continuous. The principle generalizes in a straightforward way to simultaneous fixpoints.

As an example, let us prove the following

Lemma 15.2
For a language V,

$$V \bullet V^{(*)} = V^{(*)} \bullet V$$

Proof: As our continuous predicate we choose

$$P[X] \overset{\text{def}}{\Leftrightarrow} V \bullet X = X \bullet V$$

The induction base $P[\emptyset]$ is trivial. Now assume $P[X]$. We have to show $P[C[X]]$ where $C[X] = \varepsilon \,\cup\, V \bullet X$. We calculate

$$\begin{array}{ll} & V \bullet (\varepsilon \,\cup\, V \bullet X) \\ = & V \,\cup\, V \bullet V \bullet X \\ = & \quad \{\!\!\{ \text{ by the induction hypothesis } \}\!\!\} \\ & V \,\cup\, V \bullet X \bullet V \end{array}$$

$$= (\varepsilon \cup V \bullet X) \bullet V$$

■

Using this and the properties of least fixpoints one can, moreover, show that

$$V^{(*)} = \mu Y \,.\, \varepsilon \cup V \cup V \bullet Y \bullet V$$

This way of viewing $V^{(*)}$ is important in solving the palindrome problem (see e.g. [14]).

16 JOIN

A useful derived operation is provided by a special case of the join operation as used in database theory (see e.g. [10]). Given two relations R, S, their **k-join** $R \overset{k}{\bowtie} S$ consists of all tuples that arise from "glueing" together tuples from R and from S along k common intermediate components. By our previous considerations, the language of k-tuples which are endings of tuples in R is $\mathsf{cod}_k\, R$ whereas the language of k-tuples which are beginnings of tuples in S is $\mathsf{dom}_k\, S$. Hence we define

$$R \overset{k}{\bowtie} S \overset{\text{def}}{=} \bigcup_{x \in \mathsf{cod}_k R \cap \mathsf{dom}_k S} R \circledast x \bullet x \bullet x \circledast S \tag{96}$$

Using distributivity of junction over union, this can more succinctly be expressed as [31]

$$R \overset{k}{\bowtie} S = R \circledast (\mathsf{cod}_k\, R \cap \mathsf{dom}_k\, S)^{\Delta_3} \circledast S \tag{97}$$

We can state this a bit more generally using the fact that for any x with $|x| = k$ and $x \notin \mathsf{cod}_k\, R \cap \mathsf{dom}_k\, S$ we have $R \circledast x = \emptyset$ or $x \circledast S = \emptyset$. This gives

$$R \overset{k}{\bowtie} S = R \circledast T^{\Delta_3} \circledast S \;\Leftarrow\; \mathsf{ar}\, T = k \wedge \mathsf{cod}_k\, R \cap \mathsf{dom}_k\, S \subseteq T \tag{98}$$

The join is contravariant with reversal, i.e.

$$(R \overset{k}{\bowtie} S)^{-1} = S^{-1} \overset{k}{\bowtie} R^{-1} \tag{99}$$

A special case of the join is

$$R \overset{0}{\bowtie} S = R \bullet S \tag{100}$$

As an abbreviation we introduce

$$R \bowtie S \overset{\text{def}}{=} R \overset{1}{\bowtie} S \;\Leftarrow\; \mathsf{ar}\, R, \mathsf{ar}\, S > 0 \tag{101}$$

Then

$$\mathsf{ar}\,(R \bowtie S) = \mathsf{ar}\, R + \mathsf{ar}\, S - 1 \tag{102}$$

Join and composition are closely related. To explain this we consider two binary relations R, S with $Q \overset{\text{def}}{=} \mathsf{cod}\, R \cap \mathsf{dom}\, S$:

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

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 lengths $1, 2, 3, \ldots$ in the directed graph with vertex set $\mathsf{dom}\, R \cup \mathsf{cod}\, R$ associated with R.

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

$$\begin{aligned} & R \overset{k}{\bowtie} S \\ = & \bigcup_{x \in \mathsf{cod}_k R \cap \mathsf{dom}_k S} R \,\textcircled{k}\, x \bullet x \bullet x \,\textcircled{k}\, S \\ = & \bigcup_{x \in R \cap \mathsf{dom}_k S} x \bullet x \,\textcircled{k}\, S \end{aligned}$$

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

If even $\mathsf{ar}\, R = k = \mathsf{ar}\, S$ we obtain, using (52),

$$R \overset{k}{\bowtie} S = R \cap S \tag{103}$$

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

$$R \overset{k}{\bowtie} R = R \tag{104}$$

$$x \overset{k}{\bowtie} R = R \overset{k}{\bowtie} x = \begin{cases} x & \text{if } x \in R \\ \emptyset & \text{if } x \notin R \end{cases} \tag{105}$$

$$x \overset{k}{\bowtie} y = y \overset{k}{\bowtie} x = \begin{cases} x & \text{if } x = y \\ \emptyset & \text{if } x \neq y \end{cases} \tag{106}$$

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

$$x \bowtie R \bowtie y = \begin{cases} x \bullet y & \text{if } x \bullet y \in R \\ \emptyset & \text{otherwise} \end{cases} \tag{107}$$

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

$$P \overset{k}{\bowtie} R = P^{\Delta_2} \,\textcircled{k}\, R \tag{108}$$

$$R \overset{k}{\bowtie} Q = R \,\textcircled{k}\, Q^{\Delta_2} \tag{109}$$

Domain and codomain are left and right identities of the join, resp.:

$$\mathsf{dom}_k R \overset{k}{\bowtie} R = R = R \overset{k}{\bowtie} \mathsf{cod}_k R \Leftarrow k \leq \mathsf{ar}\, R \tag{110}$$

By the associativity of junction (54) also join and junction associate:

$$(R \overset{k}{\bowtie} S) \,\textcircled{n}\, T = R \overset{k}{\bowtie} (S \,\textcircled{n}\, T) \tag{111}$$

$$R \,\textcircled{k}\, (S \overset{n}{\bowtie} T) = (R \,\textcircled{k}\, S) \overset{n}{\bowtie} T \tag{112}$$

provided $\mathsf{ar}\, S \geq k + n$.

Another useful pair of laws is

$$R \,\textcircled{k}\, (S \overset{k}{\bowtie} T) = (R \cap S) \,\textcircled{k}\, T \Leftarrow \mathsf{ar}\, R = \mathsf{ar}\, S = k \leq \mathsf{ar}\, T \tag{113}$$

$$(R \overset{k}{\bowtie} S) \,\textcircled{k}\, T = R \,\textcircled{k}\, (S \cap T) \Leftarrow \mathsf{ar}\, S = \mathsf{ar}\, T = k \leq \mathsf{ar}\, R \tag{114}$$

Moreover, also joins associate:

$$R \overset{k}{\bowtie} (S \overset{n}{\bowtie} T) = (R \overset{k}{\bowtie} S) \overset{n}{\bowtie} T \Leftarrow k = n \vee \mathsf{ar}\, S \geq k + n \tag{115}$$

Proof: We define $U \overset{\text{def}}{=} \bigcup\limits_{x\in R\cup S\cup T} \bigcup\limits_{i=1}^{|x|} x_i$. Thus U is the set of all top-level components of tuples in R, S, T. Moreover, we set $V \overset{\text{def}}{=} U^k$ and $W \overset{\text{def}}{=} U^n$. Then

$$
\begin{array}{cl}
 & R \overset{k}{\bowtie} (S \overset{n}{\bowtie} T) \\
= & \quad \{\!\!\{ \text{ by } (98) \}\!\!\} \\
 & R \textcircled{k} V^{\Delta_3} \textcircled{k} (S \textcircled{n} W^{\Delta_3} \textcircled{n} T) \\
= & \quad \{\!\!\{ \text{ by the assumption and } (54) \}\!\!\} \\
 & R \textcircled{k} (V^{\Delta_3} \textcircled{k} (S \textcircled{n} W^{\Delta_3})) \textcircled{n} T \\
= & \quad \{\!\!\{ \text{ by } (58) \}\!\!\} \\
 & R \textcircled{k} ((V^{\Delta_3} \textcircled{k} S) \textcircled{n} W^{\Delta_3}) \textcircled{n} T \\
= & \quad \{\!\!\{ \text{ by the assumption and } (54) \}\!\!\} \\
 & (R \textcircled{k} V^{\Delta_3} \textcircled{k} S) \textcircled{n} W^{\Delta_3} \textcircled{n} T \\
= & \quad \{\!\!\{ \text{ by } (98) \}\!\!\} \\
 & (R \overset{k}{\bowtie} S) \overset{n}{\bowtie} T
\end{array}
$$

■

The join also satisfies a number of further idempotence properties. Assume $\mathsf{ar}\, R = k$. Then

$$(P \overset{k}{\bowtie} R) \textcircled{k} S = (P \overset{k}{\bowtie} R) \textcircled{k} (R \overset{k}{\bowtie} S) = P \textcircled{k} (R \overset{k}{\bowtie} S) \tag{116}$$

Proof: We only prove the first equality; the proof of the second one is symmetric.

$$
\begin{array}{cl}
 & (P \overset{k}{\bowtie} R) \textcircled{k} (R \overset{k}{\bowtie} S) \\
= & (P \textcircled{k} R^{\Delta_2}) \textcircled{k} (R^{\Delta_2} \textcircled{k} S) \\
= & P \textcircled{k} R^{\Delta_2} \textcircled{k} R^{\Delta_2} \textcircled{k} S \\
= & P \textcircled{k} R^{\Delta_2} \textcircled{k} S \\
= & (P \overset{k}{\bowtie} R) \textcircled{k} S
\end{array}
$$

■

By substituting P resp. S for R we obtain, together with (104)

$$P \textcircled{k} (P \overset{k}{\bowtie} R) = P \textcircled{k} R \tag{117}$$

$$(R \overset{k}{\bowtie} Q) \textcircled{k} Q = R \textcircled{k} Q \tag{118}$$

Iterated joins can be compressed into a single one. Assume $\mathsf{ar}\, P_1 = k = \mathsf{ar}\, P_2$. Then

$$P_1 \textcircled{k} (P_2 \overset{k}{\bowtie} R) = (P_1 \cap P_2) \textcircled{k} R \tag{119}$$

$$(R \overset{k}{\bowtie} Q_1) \textcircled{k} Q_2 = R \textcircled{k} (Q_1 \cap Q_2) \tag{120}$$

$$P_1 \textcircled{k} (P_2 \overset{k}{\bowtie} R) = P_2 \textcircled{k} (P_1 \overset{k}{\bowtie} R) \tag{121}$$

$$(R \overset{k}{\bowtie} Q_1) \textcircled{k} Q_2 = (R \overset{k}{\bowtie} Q_2) \textcircled{k} Q_1 \tag{122}$$

Proof: We only prove the first equality; the proof of the second one is symmetric. The third and fourth equalities are immediate from the first and second one, resp.

$$\begin{aligned}
& P_1 \textcircled{k} (P_2 \overset{k}{\bowtie} R) \\
= \ & P_1 \textcircled{k} (P_2^{\Delta_2} \textcircled{k} R) \\
= \ & (P_1 \textcircled{k} P_2^{\Delta_2}) \textcircled{k} R \\
= \ & \quad \{\!\![\text{ by } (18{,}52) \,]\!\!\} \\
& (P_1 \cap P_2) \textcircled{k} R
\end{aligned}$$

■

Finally, a relation may be split according to a subset of its domain or codomain. Assume $P \subseteq \mathsf{dom}_k\, R, Q \subseteq \mathsf{cod}_k\, R$. Then by the distributivity of join over union we have

$$R = P \overset{k}{\bowtie} R \cup \overline{P} \overset{k}{\bowtie} R \tag{123}$$

$$R = R \overset{k}{\bowtie} Q \cup R \overset{k}{\bowtie} \overline{Q} \tag{124}$$

where $\overline{P}, \overline{Q}$ are the relative complements of P, Q w.r.t. $\mathsf{dom}_k\, R, \mathsf{cod}_k\, R$, resp.

17 CLOSURES

Consider a binary relation, i.e., a relation R with $\mathsf{ar}\, R = 2$ and let $V \overset{\text{def}}{=} \mathsf{dom}\, R \cup \mathsf{cod}\, R$. We define the **(reflexive and transitive) closure** R^* of R by

$$R^* \overset{\text{def}}{=} \mu X \in \mathcal{P}(V \bullet V) \,.\, \tau_R[X] \tag{125}$$

where

$$\tau_R[X] = V^{\Delta_2} \cup R \,;\, X \tag{126}$$

Fixpoint iteration gives

$$R^* = \bigcup_{i \in \mathbb{N}} X_i \tag{127}$$

with

$$X_0 = \emptyset \tag{128}$$

$$X_{i+1} = \tau_R[X_i] \tag{129}$$

An easy induction shows that

$$X_i = \bigcup_{j=0}^{i-1} R^j \tag{130}$$

where, as usual,

$$R^0 \overset{\text{def}}{=} V^{\Delta_2} \tag{131}$$

$$R^{i+1} \overset{\text{def}}{=} R \,;\, R^i \tag{132}$$

Let G be the directed graph associated with R, i.e., the graph with vertex set V and arcs between the vertices corresponding to the pairs in R. We have

$$x \,;\, R^i \,;\, y = \begin{cases} \varepsilon & \text{if there is a path of length } i \text{ from } x \text{ to } y \text{ in } G \\ \emptyset & \text{otherwise} \end{cases} \tag{133}$$

Likewise,

$$x \,;\, R^* \,;\, y = \begin{cases} \varepsilon & \text{if there is a path from } x \text{ to } y \text{ in } G \\ \emptyset & \text{otherwise} \end{cases} \tag{134}$$

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

$$s\,;R^*;t = \begin{cases} \varepsilon & \text{if } s \text{ and } t \text{ are connected by some path in } G \\ \emptyset & \text{otherwise} \end{cases} \tag{135}$$

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

$$R^{\bowtie} \stackrel{\text{def}}{=} \mu Y \in \mathcal{P}(V^{(*)}) \,.\, \sigma_R[Y] \tag{136}$$

where

$$\sigma_R[Y] = V \;\cup\; R \bowtie Y \tag{137}$$

Fixpoint iteration gives

$$R^{\bowtie} = \bigcup_{i \in \mathbb{N}} Y_i \tag{138}$$

with

$$Y_0 = \emptyset \tag{139}$$

$$Y_{i+1} = \sigma_R[Y_i] \tag{140}$$

An easy induction shows that

$$Y_i = \bigcup_{j=0}^{i-1} {}^{j}R \tag{141}$$

where now

$${}^{0}R \stackrel{\text{def}}{=} V \tag{142}$$

$${}^{i+1}R \stackrel{\text{def}}{=} R \bowtie {}^{i}R \tag{143}$$

The path closure consists of all finite paths in G. Hence

$$x \bowtie R^{\bowtie} \bowtie y \tag{144}$$

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

A sequence s of nodes is a cycle in G iff $s \in R^{\bowtie}\backslash V \wedge \leftarrow s \in R^{\bowtie}\backslash V$ or, equivalently, iff $s \in R^{\bowtie}\backslash V \wedge \hookrightarrow s \in R^{\bowtie}\backslash V$. Using this, it is easy to see that $(R^{\bowtie} \cap \leftarrow R^{\bowtie})\backslash V = (R^{\bowtie} \cap \hookrightarrow R^{\bowtie})\backslash V$ is precisely the set of cycles in G.

It should be noted that our definitions of closures also apply to relations of higher arity. Hence one could use them on a representation of directed graphs with labelled edges by ternary relations $R \subseteq \langle V\rangle \bullet \langle L\rangle \bullet \langle V\rangle$ where V is the set of nodes and L is the set of edge labels. Then $R^{\bowtie}$ is the set of all sequences $\langle x_1 l_1 x_2 l_2 \cdots x_n l_n x_{n+1}\rangle$ with $x_i \in V$ and $l_i \in L$ such that $\langle x_1 x_2 \cdots x_n x_{n+1}\rangle$ is a path in the graph and $\langle l_1 l_2 \cdots l_n\rangle$ is the corresponding sequence of edge labels. By contrast, R^* is the set of all sequences $\langle x l_1 l_2 \cdots l_n y\rangle$ with $x, y \in V$ and $l_i \in L$ such that there is a path in the graph from x to y and $\langle l_1 l_2 \cdots l_n\rangle$ is the sequence of edge labels on that path.

Moving away from the graph view, the path closure also is useful for general binary relations. Let e.g. $\leq$ be a partial order. Then $\leq^{\bowtie}$ is the language of all $\leq$-non-decreasing sequences. If $\leq$ is even a linear order then $\leq^{\bowtie}$ is the language of all sequences which are sorted w.r.t. $\leq$.

Note that by our definitions, for a language V of singleton tuples,

$$(\mathcal{P}(V^{(*)}),\ \cup,\ \bullet,\ .^{(*)},\ \emptyset,\ \varepsilon)$$
$$(\mathcal{P}(V \bullet V),\ \cup,\ ;,\ .^{*},\ \emptyset,\ V^{\Delta_2})$$
$$(\mathcal{P}(V^{(*)}\backslash\varepsilon),\ \cup,\ \bowtie,\ .^{\bowtie},\ \emptyset,\ V)$$

all form Kleene algebras [9].

We now prove two properties of the reflexive transitive closure which will be of use in the derivation of a reachability algorithm. To save parentheses, we define for the sequel that join binds tighter than composition which, in turn, binds tighter than union.

When computing the points reachable from some set, we can remove all paths starting in a set s if these are already covered otherwise:

Lemma 17.1
Let V be a language of singleton tuples and assume $R \subseteq V \bullet V$ and $s, t \subseteq V$. Then

$$\begin{array}{lrcl} (1) & t\,;R^* & \subseteq & s\,;R^* \cup t\,;(\overline{s} \bowtie R)^* \\ (2) & s\,;R^* & \subseteq & s \cup s\,;R\,;(\overline{s} \bowtie R)^* \end{array}$$

where $\overline{s} \overset{\text{def}}{=} V \backslash s$.

Proof: (1) by computational induction using the continuous predicate

$$P[X] \overset{\text{def}}{\Leftrightarrow} \forall s,t.\ t\,;X \subseteq s\,;R^* \cup t\,;(\overline{s} \bowtie R)^*$$

The base case $P[\emptyset]$ is immediate. For the induction step assume $P[X]$. Then

$$\begin{array}{ll} & t\,;\tau_R[X] \\ = & t\,;(V^{\Delta_2} \cup R\,;X) \\ = & t \cup t\,;R\,;X \\ \subseteq & \quad \{\!\!\mid \text{by the induction hypothesis } P[X] \text{ specialized to } t\,;R \mid\!\!\} \\ & t \cup s\,;R^* \cup t\,;R\,;(\overline{s} \bowtie R)^* \\ = & s\,;R^* \cup t \cup t\,;(s \bowtie R \cup \overline{s} \bowtie R)\,;(\overline{s} \bowtie R)^* \\ = & s\,;R^* \cup t \cup t\,;s \bowtie R\,;(\overline{s} \bowtie R)^* \cup t\,;\overline{s} \bowtie R\,;(\overline{s} \bowtie R)^* \\ = & \quad \{\!\!\mid \text{by (113)} \mid\!\!\} \\ & s\,;R^* \cup t \cup (t \cap s)\,;R\,;(\overline{s} \bowtie R)^* \cup t\,;\overline{s} \bowtie R\,;(\overline{s} \bowtie R)^* \\ = & \quad \{\!\!\mid \text{by } t \cap s \subseteq s,\ \overline{s} \bowtie R \subseteq R, \text{ and monotonicity} \mid\!\!\} \\ & s\,;R^* \cup t \cup t\,;\overline{s} \bowtie R\,;(\overline{s} \bowtie R)^* \\ = & s\,;R^* \cup t\,;(\overline{s} \bowtie R)^* \end{array}$$

(2) by computational induction using the continuous predicate

$$Q[X] \overset{\text{def}}{\Leftrightarrow} \forall s.\ s\,;X \subseteq s \cup s\,;R\,;(\overline{s} \bowtie R)^*$$

The base case $Q[\emptyset]$ is immediate. For the induction step assume $Q[X]$. Then

$$\begin{array}{ll} & s\,;\tau_R[X] \\ = & s\,;(V^{\Delta_2} \cup R\,;X) \\ = & s \cup s\,;R\,;X \\ \subseteq & \quad \{\!\!\mid \text{by the induction hypothesis } Q[X] \text{ specialized to } s\,;R \mid\!\!\} \\ & s \cup s\,;R \cup s\,;R\,;R\,;(\overline{s} \bowtie R)^* \\ = & s \cup s\,;R \cup s\,;R\,;(s \bowtie R \cup \overline{s} \bowtie R)\,;(\overline{s} \bowtie R)^* \\ = & s \cup s\,;R \cup s\,;R\,;s \bowtie R\,;(\overline{s} \bowtie R)^* \cup s\,;R\,;\overline{s} \bowtie R\,;(\overline{s} \bowtie R)^* \\ = & s \cup s\,;R\,;(\overline{s} \bowtie R)^* \cup s\,;R\,;s \bowtie R\,;(\overline{s} \bowtie R)^* \\ = & \quad \{\!\!\mid \text{by (113)} \mid\!\!\} \\ & s \cup s\,;R\,;(\overline{s} \bowtie R)^* \cup (s\,;R \cap s)\,;R\,;(\overline{s} \bowtie R)^* \\ = & \quad \{\!\!\mid \text{by monotonicity, since } s\,;R \cap s \subseteq s \mid\!\!\} \end{array}$$

$$s \cup s\,;R\,;(\overline{s} \bowtie R)^*$$

■

Corollary 17.2

$$\begin{array}{lll} (1)\ s\,;R^* \cup t\,;R^* & = & s\,;R^* \cup t\,;(\overline{s} \bowtie R)^* \\ (2)\ s\,;R^* & = & s \cup s\,;R\,;(\overline{s} \bowtie R)^* \end{array}$$

Proof: Monotonicity shows the ($\supseteq$) parts; ($\subseteq$) follows from the above lemma. ■

Note that analogous properties do not hold for the path closure, since $R^{\bowtie}$ records all intermediate points of the paths, whereas R^* "forgets" these.

18 EXAMPLE: A SIMPLE REACHABILITY ALGORITHM

We consider the following problem: Given a directed graph, represented by a binary relation $R \subseteq V \bullet V$ over the set V of singleton tuples of vertices, and a subset $s \subseteq V$, compute the set of vertices reachable from paths starting in s. Hence we define (omitting type information)

$$reach \overset{\text{def}}{=} \lambda s\,.\,s\,;R^* \tag{145}$$

The aim is to derive a recursive variant of *reach* from this specification. A termination case is given by

$$\{\emptyset\}\,;reach = \emptyset\,;R^* = \emptyset \tag{146}$$

Another obvious idea is to use the fixpoint property

$$R^* = V^{\Delta_2} \cup R\,;R^* \tag{147}$$

of the closure. This gives

$$\begin{array}{ll} & \{s\}\,;reach \\ = & s\,;R^* \\ = & s\,;(V^{\Delta_2} \cup R\,;R^*) \\ = & s\,;V^{\Delta_2} \cup s\,;(R\,;R^*) \\ = & s \cup (s\,;R)\,;R^* \\ = & s \cup \{s\,;R\}\,;reach \end{array}$$

However, since there may be cycles in R, there is no guarantee for termination for this recursion. The common technique to avoid this is to keep track of the vertices already visited. So we embed *reach* into a function *re* with a second parameter t which is the set of points already visited. Then we need no longer consider paths starting in t. This is expressed by

$$re \overset{\text{def}}{=} \lambda t\,.\,\lambda s\,.\,t \cup s\,;(\overline{t} \bowtie R)^* \tag{148}$$

where $\overline{t} = V\backslash t$. We have the embedding

$$\{s\}\,;reach = \{s\}\,;(\{\emptyset\}\,;re) \tag{149}$$

Then,

$$\{\emptyset\}\,;(\{t\}\,;re) = t \tag{150}$$

so that we have a termination case. For $s \neq \emptyset$ we have $s = x \cup s\backslash x$ for all $x \in s$. We calculate

$$\begin{array}{ll} & \{s\}\,;(\{t\}\,;re) \\ = & t \cup s\,;(\overline{t} \bowtie R)^* \\ = & t \cup (x \cup s\backslash x)\,;(\overline{t} \bowtie R)^* \end{array}$$

$$
\begin{array}{ll}
= & t \cup x ; (\bar{t} \bowtie R)^* \cup (s \backslash x) ; (\bar{t} \bowtie R)^* \\
= & \quad \{\![\text{ by Corollary 17.2 }]\!\} \\
 & t \cup x \cup x ; (\bar{t} \bowtie R) ; (\overline{x} \bowtie \bar{t} \bowtie R)^* \cup (s \backslash x) ; (\overline{x} \bowtie \bar{t} \bowtie R)^* \\
= & t \cup x \cup (x ; (\bar{t} \bowtie R) \cup s \backslash x) ; (\overline{x} \bowtie \bar{t} \bowtie R)^* \\
= & t \cup x \cup (x ; (\bar{t} \bowtie R) \cup s \backslash x) ; (\overline{x} \cap \bar{t}) \bowtie R)^* \\
= & t \cup x \cup (x \cap \bar{t}) ; R \cup s \backslash x) ; (\overline{t \cup x}) \bowtie R)^* \\
= & t \cup x \cup ((x \backslash t) ; R \cup s \backslash x) ; (\overline{t \cup x}) \bowtie R)^* \\
= & \{(x \backslash t) ; R \cup s \backslash x\} ; (\{x \cup t\} ; re)
\end{array}
$$

Altogether,

$$
re = \lambda t \,.\, \lambda s \,.\, s = \emptyset \rightarrow \left\{ \begin{array}{l} t \\ \bigcup\limits_{x \in s} \{(x \backslash t) ; R \cup s \backslash x\} ; (\{x \cup t\} ; re) \end{array} \right\} \tag{151}
$$

Note that this is a tail recursion. Termination is guaranteed, since for each recursive call either the first parameter decreases or the second one increases.

19 CONCLUSION

The relational approach offers a surprisingly simple, yet very powerful framework for calculational program development. We have given only one example in the present paper; however, further (and much more intricate) ones can be obtained by transcribing the developments in [25,4,27] into relational notation. It is to be hoped that the incorporation of the basic operations of formal language theory makes the framework also suitable for the treatment of parsing and pattern recognition algorithms.

A lot remains to be done. One has to define a suitable notion of refinement between relational expressions; a crude approximation, of course, is the superset relation which describes the reduction of nondeterminacy but also allows decreasing the "definedness". The main problem is to find a refinement relation w.r.t. which all operators are monotonic. A promising candidate is discussed in [14]. Then one needs to show, as in [5], that the unfold/fold transformation strategy also is correct when refinement steps are used instead of equality steps only.

Next, it has to be investigated how the results of [1] can be expressed in our framework, so that larger development steps can be taken.

Finally, since data types are "first-class citizens" in this framework, one should investigate the possibilities of data type transformations using the rules of the relational operators.

Acknowledgments

I am grateful to H. Partsch and D. Smith for encouraging me to write this paper, and to W. Dosch, H. Ehler, F. Regensburger, and M. Russling for providing many valuable comments. R. Bird has pointed out a simplification in the derivation of the reachability algorithm. Particular thanks are due to J. van der Woude for a number of suggestions which considerably improved the presentation.

20 REFERENCES

1. R.C. Backhouse, P.J. de Bruijn, G. Malcolm, E. Voermans, J. van der Woude: Relational catamorphisms. This volume
2. 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
3. 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)
4. U. Berger, W. Meixner, B. Möller: Calculating a garbage collector. In: M. Broy, M. Wirsing (Hrsg.): Methodik des Programmierens. Fakultät für Mathematik und Informatik, Universität Passau, MIP-8915, 1989, 1–52. Also in: M. Broy, M. Wirsing (eds.): Programming methodology — The CIP approach. Berlin: Springer (to appear)
5. R. Berghammer, H. Ehler, B. Möller: On the refinement of nondeterministic recursive routines by transformation. In: M. Broy, C.B. Jones (eds.): Programming concepts and methods. Amsterdam: North-Holland 1990, 53–71
6. 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
7. R. Bird, L. Meertens, D. Wile: A common basis for algorithmic specification and development. Document No. 477 ARK-3 of IFIP Working Group 2.1, 1985
8. R.M. Burstall, J. Darlington: A transformation system for developing recursive programs. J. ACM **24**, 44–67 (1977)
9. J.H. Conway: Regular algebra and finite machines. London: Chapman and Hall 1971
10. C.J. Date: An introduction to database systems. Vol. I, 4th edition. Reading, Mass.: Addison-Wesley 1988
11. J. Desharnais, N.H. Madhavji: Abstract relational specifications. In: M. Broy, C.B. Jones (eds.): Programming concepts and methods. Amsterdam: North-Holland 1990, 267–284
12. E.W. Dijkstra: Guarded commands, non-determinacy, and formal derivation of programs. Commun. ACM **18**, 453–457 (1985)
13. S. Ginsburg: The mathematical theory of context-free languages. New York: McGraw-Hill 1966
14. A.M. Haeberer, P.A.S. Veloso: Partial relations for program derivation — Adequacy, inevitability and expressiveness. This volume
15. M. Hennessy: The semantics of call-by-value and call-by-name in a non-deterministic environment. SIAM J. Comp. **1**, 67-84 (1980)
16. M. Hennessy, E.A. Ashcroft: The semantics of non-determinism. In S. Michaelson, R. Milner (eds.): Automata, languages and programming. Edinburgh: Edinburgh

University Press 1976, 478–493

17. S.C. Kleene: Introduction to metamathematics. New York: van Nostrand 1952
18. B. Knaster: Un théorème sur les fonctions d'ensembles. Ann. Soc. Pol. Math. **6**, 133–134 (1928)
19. P. Martin-Löf: Constructive mathematics and computer programming. In L.J. Cohen et al. (eds.): Logic, methodology and philosophy of science VI. Amsterdam: North-Holland 1982, 153–175
20. J. McCarthy: A basis for a mathematical theory of computation. In: P. Braffort, D. Hirschberg (eds.): Computer programming and formal systems. Amsterdam: North-Holland 1963, 33–70
21. 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
22. A. Mili: A relational approach to the design of deterministic programs. Acta Informatica **20**, 315–329 (1983)
23. J.C. Mitchell: Type systems for programming languages. In J. van Leeuwen (ed.): Handbook of theoretical computer science. Volume B: Formal models and semantics. Amsterdam: Elsevier 1990, 365–458
24. 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
25. B. Möller: Some applications of pointer algebra. In: M. Broy (ed.): Programming and mathematical method. Proc. International Summer School Marktoberdorf 1990. Berlin: Springer (to appear)
26. P.D. Mosses: Unified algebras and modules. Proc. ACM Conference on Principles of Programming Languages 1989
27. P. Pepper, B. Möller: Programming with (finite) mappings. In: M. Broy (ed.): Informatik im Kreuzungspunkt von Numerischer Mathematik, Rechnerentwurf, Programmierung, Algebra und Logik. Berlin: Springer (to appear)
28. G. Schmidt, T. Ströhlein: Relationen und Graphen. Springer 1989
29. A. Tarski: On the calculus of relations. J. Symbolic Logic **6**, 73–89 (1941)
30. A. Tarski: A lattice-theoretical fixpoint theorem and its applications. Pacific J. Math. **5**, 285–309 (1955)
31. J. van der Woude: Personal communication