RELATIONAL ALGEBRAIC SEMANTICS OF DETERMINISTIC AND NONDETERMINISTIC PROGRAMS *

R. BERGHAMMER and H. ZIERER


Communicated by M. Nivat
Received April 1985
Revised December 1985

Abstract. Abstract relational algebra is proposed as a practical means to describe the denotational semantics of programming languages. We apply this method of semantics description to a functional language and demonstrate the usefulness of this approach by some examples. In particular, we prove the correctness of a program transformation rule within our calculus.

Then, the deterministic language DFP is extended to a nondeterministic functional language NFP in order to investigate three kinds of nondeterminism, viz. erratic, angelic, and demonic nondeterminism. We provide a relational algebraic semantics for each kind of nondeterminism. These three different semantics are not obtained by using three different orderings on certain domains, but by using three different interpretations on the same ordered domain. The alternative kinds of nondeterminism are then compared and an illustrative example is given.

Finally, we characterize the natural numbers with relational algebraic means. The uniform description of domains and programs makes it possible to prove the termination of 'concrete' deterministic and nondeterministic programs by induction.

1. Introduction

In this paper we propose abstract relational algebra as a practical means to describe the denotational semantics of programming languages and to prove the correctness of program transformation rules. We apply this calculus not only to a deterministic language but also to a nondeterministic one. Using a (monomorphic) relational algebraic characterization of data structures, it is also possible to consider concrete programs and to prove termination by induction.

A relational algebra is an algebraic structure such that the usual identities valid for 'concrete' relations on sets can be deduced from the axioms. The calculus of relations is due to Tarski and his collaborators (cf. [5, 17]). However, they restricted themselves to 'quadratic' relations on one set. An extension to the case of relations on several sets was given by Schmidt and Ströhlein in [13, 15]. In [16] the axioms of a relational algebra are discussed in detail.

Applying the relational calculus to the mathematical theory of programs has several advantages: within this framework we need not cope with extended domains.

* This work was partially sponsored by the Sonderforschungsbereich 49, Programmiertechnik, München.
and with the additional 'bottom'-element. Therefore, it is very easy to handle nondefinedness. As relations are ordered by inclusion, a fixed-point approach to semantics suggests itself and the Scott induction rule may be used. Moreover, the 'linear' nature of relational algebra facilitates investigations of programs (program schemes) and proofs of their properties.

As an example language we choose a dialect of Backus' functional language FP (cf. [1]). We concentrate on the control structure of FP and neglect the object level of FP. Thus, our variant DFP of FP is a scheme language. A concrete program can be obtained by providing an interpretation for the symbols of the scheme.

For the deterministic language DFP the semantics given is quite similar to the relational calculus of De Bakker and De Roever [6]. Like De Roever [8], we also consider 'polyadic' recursive program schemes and we slightly generalize his approach by allowing partial predicates.

For program transformation systems and in connection with parallelism, nondeterminism is very useful (cf. [2, 4, 7]). Therefore, we extend DFP to the nondeterministic scheme language NFP by introducing a new functional combinator for nondeterministic branching.

From the literature three kinds of nondeterminism are known: angelic, demonic, and erratic nondeterminism. We define a mathematical semantics for each kind of nondeterminism. Thus, we are able to compare these three different concepts. The obvious extension of the deterministic relational semantics to nondeterminism only leads to angelic nondeterminism. To describe the semantics of erratic (and demonic) nondeterminism, we use pairs \((B, d)\) of relations where \(B\) corresponds to the 'breadth' of a functional form and \(d\) to its 'definedness', cf. [4]. On such pairs we introduce the well-known Egli-Milner-ordering, but we define it by basing it on the usual inclusion ordering of relations without requiring an additional 'bottom'-element as in [7]. Thus, the advantages of the relational calculus are fully preserved for nondeterministic semantics.

We do not use three different ordered domains (say Egli-Milner-, Smyth-, Hoare-powerdomain) to describe the three kinds of nondeterministic semantics, but we obtain the different semantics by using three interpretations on the same ordered domain (by the Egli-Milner-ordering). Thus, it is easier to apply the 'breadth'/'definedness' formalism and relational algebra to all three kinds of nondeterminism.

Relational algebra seems most appropriate to describe the semantics of functional languages, but also procedural or flowchart languages may be dealt with within this calculus (cf. [13, 20]). Even data structures may be characterized using relational algebraic means (cf. [6, 14]). Because of this unified description of data structures and programs we are able to prove the termination of programs by computational induction.

The paper is organized as follows. In Section 2, we shall briefly explain the basic concepts of relational algebra. Furthermore, we shall give a monomorphic characterization of the domain \(B\) of truth values and of the direct product of domains
with relational algebraic means (cf. [15, 20]). In contrast to [8], the characterization of the direct product will be given by a first order construction.

In Section 3, the syntax and semantics of the deterministic functional programming language DFP will be defined. The last part of this section contains examples of program schemes. We shall prove several algebraic laws and a transformation rule for transforming a recursive program scheme into a repetitive one.

In Section 4, we shall extend DFP to the nondeterministic functional language NFP. After defining the syntax of NFP, we shall give an erratic semantics. In the same way (also using 'breadth' and 'definedness'), an angelic and a demonic semantics of NFP will be introduced. These two kinds of nondeterministic semantics can be completely characterized by their breadth parts because the 'definedness' equals the domain of the 'breadth'. Nevertheless, we shall use the different kinds of 'breadth'/'definedness' semantics to compare the three kinds of nondeterminism.

In the last section, we shall define the natural numbers with relational algebraic means and show that this characterization is monomorphic. As an example we shall prove the termination of a recursive program over the natural numbers using Scott induction.

2. Relational algebraic preliminaries

This section deals with the fundamental concepts of an abstract relational algebra. We also define relations fulfilling certain properties. In the heterogeneous case we look for 'quasi functional' properties such as uniqueness, totality and so on. Finally, we introduce the concepts of homomorphism and isomorphism.

2.1. Relational algebra

The axiomatization of a relational algebra is due to Chin and Tarski [5]. However, they considered only 'quadratic' relations on one set. For a comprehensive explanation of the basic concepts of a partial relational algebra we refer to [13, 16], where more details are presented.

A relational algebra is an algebraic structure \((\mathcal{R}, \lor, \land, \neg, \subseteq, \subseteq^T, \cdot)\) over a nonempty set \(\mathcal{R}\) of elements, called relations. Every relation \(R \in \mathcal{R}\) belongs to a subset \(\mathcal{R}_R\) of \(\mathcal{R}\) such that the following conditions are fulfilled:
- \((\mathcal{R}_R, \lor, \land, \neg, \subseteq)\) is a complete atomistic Boolean algebra. As usual, the ordering \(\subseteq\) between relations is called inclusion. With \(0\) and \(L\) we denote the null element and the universal element of \(\mathcal{R}_R\), respectively.
- For every relation \(R\), there exist a transposed relation \(R^T\) and the products \(R^T R\) and \(RR^T\).
- Multiplication is associative and the existence of a product \(RS\) implies that \(QS\) is defined for all relations \(Q \in \mathcal{R}_R\). There exist right and left identities for every set \(\mathcal{R}_R\) of relations, which, for simplicity, are all denoted by \(I\).
Finally, the Dedekind rule
\[(QR \land S) \subseteq (Q \land SR^T)(R \land Q^TS)\]
holds whenever one of the three parenthetical expressions is defined.

If \(R^T \subseteq R\) holds for a relation \(R\), then \(R\) is called \textit{homogeneous} (or quadratic). All the well-known rules for composition of relations hold in a relational algebra. They may be deduced from the axioms. As first examples we cite \(0^T = 0\), \(L^T = L\), and \(I^T = I\), where possibly different null and universal relations are denoted by the same letter.

Furthermore, we note:
\[
\begin{align*}
(R^T)^T &= R, \\
RS^T &= (SR)^T, \\
R^TS^T &= (SR)^T, \\
R \subseteq S &\Rightarrow QR \subseteq QS, \\
R \subseteq S &\Rightarrow RQ \subseteq SQ, \\
R(S \land Q) &\subseteq RS \land RQ, \\
R(S \lor Q) &= RS \lor RQ, \\
(R \land S)^T &= R^T \land S^T, \\
(R \lor S)^T &= R^T \lor S^T.
\end{align*}
\]

In addition, we have the so-called \textit{Schröder rule}
\[RS \subseteq Q \Leftrightarrow R^TQ \subseteq S \Leftrightarrow Q^TS \subseteq R.\]

These two equivalences are equivalent to the Dedekind rule.

\textbf{2.1.1. Special heterogeneous relations}

A relation \(R\) is called \textit{unique} if \(R^TR \subseteq I\) or, equivalently, if \(R\bar{I} \subseteq \bar{R}\). If one of the three equivalent conditions \(I \subseteq RR^T\), \(L = RL\), and \(\bar{R} \subseteq R\bar{I}\) is fulfilled, \(R\) will be called \textit{total}. Thus, \textit{mappings} (or \textit{functions}), i.e., total and unique relations, are characterized by \(R \bar{I} = \bar{R}\). If \(R\) is unique, then \(R(S \land Q) = RS \land RQ\) and \(R\bar{S} \subseteq R\bar{S}\) and \((Q \land SR^T)R = QR \land S\); if \(R\) is total, then \(R\bar{S} \subseteq R\bar{S}\). Therefore, for a function \(R\) we have \(R\bar{S} = R\bar{S}\) for any relation \(S\). A relation \(R\) is \textit{injective}, if \(R^T\) is unique; \(R\) is called \textit{surjective}, if \(R^T\) is total. Note that these properties are defined for arbitrary relations, not only, as usually, for functions.

\textbf{2.1.2. Sets and points}

A relation \(r\) with \(r = rL\) is called \textit{row constant}. If we consider a (concrete) relation \(r\) as a Boolean matrix \(r \in \mathbb{B}^{X \times Y}\), this condition means: whatever set \(Z\) and universal relation \(L \in \mathbb{B}^{Y \times Z}\) we choose, an element \(x \in X\) is either in relation \(rL\) to none of the elements \(z \in Z\) or to all elements \(z \in Z\). Relations of this kind may be considered as \textit{subsets} of \(X\), \textit{predicates} on \(X\), or \textit{vectors}. A bijective vector \(r = rL\) therefore corresponds to an element of \(X\) and is called a \textit{point}. The transpose \(r^T\) of a point \(r\) is unique and total. So it corresponds to a 0-ary function. For a point \(r\) the equation \(r^T r = Lr^T rL = L\) holds. For vectors we have the following theorem.
Theorem 2.1. Let $Q$, $R$, and $S$ be relations. Then

$$ (QL \land R)S = QL \land RS, \quad Q(R \land SL) = (Q \land (SL)^T)R. $$

Proofs can be found in [13].

2.1.3. Homomorphisms

Let $R$ and $S$ be (heterogeneous) relations. A pair $(\psi, \phi)$ of relations is called a homomorphism from $R$ to $S$ if $\psi$ and $\phi$ are functions (in the relational sense) and $R \subseteq \psi S \phi^T$ holds. An equivalent version of this postulate is $R \phi \subseteq \psi S$. This, in turn, is equivalent to $\psi^T R \phi \subseteq S$ and to $\psi^T R \subseteq S \phi^T$. If, in addition, $(\psi^T, \phi^T)$ is a homomorphism from $S$ to $R$, then $(\psi, \phi)$ is called an isomorphism. Therefore, an isomorphism between two relations $R$ and $S$ is characterized by two bijective functions $\psi$ and $\phi$, fulfilling $R = \psi S \phi^T$ or equivalently $R \phi = \psi S$. Clearly, the composition $(\psi_1, \psi_2, \phi_1, \phi_2)$ of two homomorphisms (isomorphisms) $(\psi_1, \phi_1)$ and $(\psi_2, \phi_2)$ is also a homomorphism (isomorphism). If $R$ and $S$ are homogeneous relations, we briefly call $\phi$ a homomorphism (isomorphism) if $(\phi, \phi)$ is a homomorphism (isomorphism) from $R$ to $S$.

At the end of this section, let us return to the algebraic structure of a relational algebra. Naturally, the most important model of this structure is the set of concrete relations between several sets, the set of all Boolean matrices corresponding to these relations respectively. In general, however, a structure fulfilling the axioms of a relational algebra need not necessarily equal the relations on sets. Examples can be found in [15].

2.2. Relational domain constructions

As a primitive domain we need the domain $\mathbb{B}$ of truth values. Following [20], this domain can be characterized by a triple $(T, F, I_2)$ of relations, such that

1. $T$ and $F$ are surjective vectors;
2. $T \land F = 0$;
3. $I_2^T I_2 \subseteq I_2$.

This characterization is monomorphic. Furthermore, $I_2 = TT^T \lor FF^T$, $T = \bar{F}$ and $T$, $F$ are points. Hence, we call the pair $(T, F)$ a characterization of $\mathbb{B}$.

Nonprimitive domains can be constructed by direct products and direct sums. In this paper, we only need direct products. In [8], a relational characterization of the associative direct product is given. The following definition of the direct product is taken from [15] and [20].

Definition 2.2. The pair $(\pi_1, \pi_2)$ of relations is called a two-fold direct product if

1. $\pi_1^T \pi_1 = I$,
2. $\pi_2^T \pi_2 = I$,
3. $\pi_1 \pi_1^T \land \pi_2 \pi_2^T = I$,
4. $\pi_1 \pi_2^T = I$.

It can be shown that this characterization is monomorphic. This allows a (monomorphic) characterization of the $n$-fold direct product, $n \geq 3$. 

Definition 2.3. Let $\pi_j$, $1 \leq j \leq n$, be a set of relations. We call $(\pi_1, \ldots, \pi_n)$ an $n$-fold direct product if there are relations $\rho, \beta_1, \ldots, \beta_{n-1}$ such that

1. $(\beta_1, \ldots, \beta_{n-1})$ is a $(n-1)$-fold direct product,
2. $\rho \beta_k = \pi_k$, $1 \leq k \leq n - 1$,
3. $(\rho, \pi_n)$ is a two-fold direct product.

The relations $\pi_j$ are called the projection relations of the direct product.

The following properties of direct products can easily be shown.

Theorem 2.4. Let $(\pi_1, \ldots, \pi_n)$ be a direct product.

(a) $\pi_k \pi_k = I$ for $1 \leq k \leq n$,
(b) $\land_{k=1}^{n} \pi_k \pi_k^T = I$,
(c) $\pi_j \pi_k = L$ for $1 \leq j, k \leq n$ and $j \neq k$.

An important construction is tupeling of relations.

Definition 2.5. Let $P = (\pi_1, \ldots, \pi_n)$ be a direct product and $A_k$, $1 \leq k \leq n$, relations.

Then we define $[A_1, \ldots, A_n]_P = \land_{k=1}^{n} A_k \pi_k^T$ the tupeling of the $A_k$ w.r.t. the direct product $P$.

Mostly, we drop the subscript $P$ and write $[A_1, \ldots, A_n]$ instead of $[A_1, \ldots, A_n]_P$.

The next theorems are a collection of properties subsequently used. Their proofs are easy.

Theorem 2.6. Assume $(\pi_1, \ldots, \pi_n)$ to be a direct product. If $A_1, \ldots, A_n$ are unique (mappings), then $[A_1, \ldots, A_n]$ is (a) unique (mapping), as well.

From distributivity, we obtain

$$[A_1, \ldots, A_k^{(1)} \lor A_k^{(2)}, \ldots, A_n] = \lor_{j=1}^{2} [A_1, \ldots, A_k^{(j)}, \ldots, A_n],$$

$$[A_1, \ldots, A_k^{(1)} \land A_k^{(2)}, \ldots, A_n] = \land_{j=1}^{2} [A_1, \ldots, A_k^{(j)}, \ldots, A_n].$$

Clearly, $B[A_1, \ldots, A_n] = [BA_1, \ldots, BA_n]$ if $B$ is unique. In the sequel, the following theorem will be often used.

Theorem 2.7. Let $(\pi_1, \pi_2)$ be a direct product. Then

$$[A_1, A_2][B_1, B_2]^T = A_1 B_1^T \land A_2 B_2^T$$

if $A_j$, $1 \leq j \leq 2$, are both unique or $B_j$, $1 \leq j \leq 2$, are both unique.\(^1\)

Proof. Let $A_1$ and $A_2$ be unique. Then

$$[A_1, A_2][B_1, B_2]^T$$

$$= [A_1, A_2] \pi_1 B_1^T \land [A_1, A_2] \pi_2 B_2^T$$ (as $[A_1, A_2]$ is unique)

$$= (A_1 \land A_2 L) B_1^T \land (A_1 L \land A_2) B_2^T$$ (cf. Section 2.1.1)

\(^1\) We assume that both tupelings are defined w.r.t. $(\pi_1, \pi_2)$. 
Relational algebraic semantics of programs

\[ A_1 B_1^T \land A_2 L \land A_1 L \land A_2 B_2^T \] (Theorem 2.1)

\[ A_1 B_1^T \land A_2 B_2^T \]

as \( A_1 B_1^T \subseteq A_1 L \). In the case that \( B_1, B_2 \) are unique, the proof follows by transposition. □

In applications, we often use \( B_j^T = C_j \pi_j^T \). In this case we obtain for unique \( A_1, A_2 \)

\[ [A_1, A_2][\pi_1 C_1, \pi_2 C_2] = [A_1, A_2][\pi_1 C_1^T, \pi_2 C_2^T]^T = [A_1 C_1, A_2 C_2]. \]

The extension of Theorem 2.7 to the \( n \)-fold direct product is obvious.

3. The deterministic functional language DFP

In this section, a simple deterministic functional programming language will be introduced. This language DFP will be defined by giving its syntax and its semantics. For simplicity, a program in DFP consists of only one definition and a functional form in which the routine identifier of the definition may occur. Our approach can easily be extended to the case of several definitions using an environment in the description of the semantics.

3.1. Syntax

We start with some given countable and disjoint sets \( F \) and \( R \) of function symbols and routine identifiers, respectively. Furthermore, let \( P \) be a countable set of projection function symbols.

Definition 3.1. The set \( \mathcal{F}_D \) of (deterministic) functional forms over \( F \) and \( R \) is defined as the least set having the following properties

(a) all symbols \( f \in F \) are functional forms;
(b) all identifiers \( r \in R \) are functional forms;
(c) all projection function symbols \( p \in P \) are functional forms;
(d) if \( t_j, 1 \leq j \leq 3 \), are functional forms, then the condition \( (t_1 \rightarrow t_2; t_3) \) is a functional form, too;
(e) if \( t_j, 1 \leq j \leq k \), \( k \geq 2 \), are functional forms, then the construction \( [t_1, \ldots, t_k] \) is a functional form, too;
(f) if \( t_1, t_2 \) are functional forms, then the (sequential) composition \( (t_1 \circ t_2) \) is a functional form, too.

We will frequently drop parentheses when ambiguities cannot occur.

As the set \( \mathcal{F}_D \) is inductively defined, we can define a relation ‘occurs free’ on \( \mathcal{F}_D \) and \( R \) in the usual way. The set of routine identifiers occurring free in the functional form \( t \) is denoted by \( \text{FRE}(t) \).
**Definition 3.2.** Let $t_1, t_2 \in \mathcal{F}_D$ be functional forms and $r \in R$ a routine identifier such that $\text{FRE}(t_1) \subseteq \{r\}$ and $\text{FRE}(t_2) \subseteq \{r\}$. Then we call $\text{def } r \leftarrow t_1$ a (functional) definition and the pair $(\text{def } r \leftarrow t_1 | t_2)$ a (deterministic) functional program. By $\mathcal{P}_D$ we denote the set of all functional programs.

### 3.2. Semantics

In this section, we define a mathematical semantics for functional forms and functional programs. This semantics is given as the least fixed point of a (partial) mapping on relations. Therefore, we assume to have a relational algebra $\mathcal{R}$, being 'big enough' to allow the following constructs. We denote the subset of unique relations of $\mathcal{R}$ by the symbol $\mathcal{U}$.

The first step is the interpretation of the set $F$ of function symbols of the language DFP.

**Definition 3.3.** An interpretation $i : F \rightarrow \mathcal{U}$ of the basis of DFP is a mapping associating a unique relation with every function symbol.

Now, the semantics of a functional form $t$ is a mapping on relations which describes the interpretation of $t$ depending on the interpretation of the routine identifier which occurs in $t$.

**Definition 3.4.** The semantic functional $E : \mathcal{F}_D \rightarrow (\mathcal{R} \rightarrow \mathcal{R})$ is inductively defined by the structure of $t \in \mathcal{F}_D$:

- (a) if $t$ is a function symbol $f \in F$, then $E[f](X) = i(f)$;
- (b) if $t$ is a routine identifier $r \in R$, then $E[r](X) = X$;
- (c) if $t$ is a projection function symbol $p \in P$, then $E[p](X) = \pi_j$, where $\pi_j$ is a projection relation from an appropriate direct product $(\pi_1, \ldots, \pi_k)$, $k \geq j$;
- (d) if $t$ is a condition $t_1 \rightarrow t_2 ; t_3$, then
  \[ E[t_1 \rightarrow t_2 ; t_3](X) = (E[t_1](X) T L \land E[t_2](X)) \lor (E[t_1](X) F L \land E[t_3](X)), \]
  where $(T, F)$ is a characterization of the data type $\mathbb{B}$;
- (e) if $t$ is a construction $[t_1, \ldots, t_k]$, then
  \[ E[[t_1, \ldots, t_k]](X) = \land_{j=1}^k (E[t_j](X) \pi_j), \]
  where $(\pi_1, \ldots, \pi_k)$ is a direct product such that the conjunction exists;
- (f) if $t$ is a composition $t_1 \circ t_2$, then
  \[ E[t_1 \circ t_2](X) = E[t_2](X)E[t_1](X). \]

Definition 3.4 implies that construction and composition are strict (or $\perp$-preserving in the terminology of [1]). The condition is—as usual—strict in its first argument but nonstrict in its second and third argument.
As DFP is a deterministic language, we want to consider only unique relations. Therefore, we have to show that \( E[t] \) maps unique relations into unique relations (for every functional form \( t \)).

This is done by induction on the structure of \( t \). The induction basis is trivial as interpretations of function symbols and of projection function symbols are unique. The induction step is also rather simple. Tupeling and multiplication of relations preserve uniqueness. If \( t \) is a condition, we use the induction hypothesis and the inclusion \( T^T F \subseteq 0 \) which follows from \( T \land F = 0 \) (cf. Section 2.2).

The following result is essential since later on it allows to give a fixed-point semantics for functional programs.

**Theorem 3.5.** For every \( t \in \mathcal{F}_D \) the mapping \( E[t] : \mathcal{R} \to \mathcal{R} \) is monotonic.

**Proof.** The proof is done by structural induction on the functional form \( t \). Let \( X \subseteq Y \).

If \( t \) is a function symbol or a projection function symbol, \( E[t](X) \) does not depend on \( X \). Therefore, \( E[t](X) = E[t](Y) \).

If \( t \) is a routine identifier \( r \), then \( E[r](X) = X \subseteq Y = E[r](Y) \).

The proof of the assertion for composed functional forms is also quite easy, e.g.,

\[
E[t_1 \circ t_2](X) = E[t_2](X)E[t_1](X)
\]

\[
\subseteq E[t_2](Y)E[t_1](Y) \quad \text{(by induction hypothesis)}
\]

\[
= E[t_1 \circ t_2](Y). \quad \square
\]

Let \( R \in \mathcal{R}_R \) be a relation and \( \tau : \mathcal{R} \to \mathcal{R} \) a monotonic mapping such that \( \tau(R) \) exists. Then \( \tau(S) \) exists for all \( S \in \mathcal{R}_R \), too. Therefore, \( \tau \) is total on the subset \( \mathcal{R}_R \).

As this set forms a complete lattice, Tarski's fixed-point theorem (cf. [18]) shows that \( \tau \) possesses a least fixed point \( \mu_\tau \) in \( \mathcal{R}_R \),

\[
\mu_\tau = \inf\{S \in \mathcal{R}_R : \tau(S) \subseteq S\}.
\]

**Definition 3.6.** The semantic functional \( M : \mathcal{P}_D \to \mathcal{R} \) assigns a relation to each functional program. The semantics of the program \( (\text{def } r \Leftarrow t_1 | t_2) \) is defined by

\[
M[\text{def } r \Leftarrow t_1 | t_2] = E[t_2](\mu_{E[t_1]}).
\]

For every relation \( R \), the set \( \mathcal{U} \cap \mathcal{R}_R \) is a complete partial ordering (cpo, cf. [9]). Therefore, a monotonic mapping from \( \mathcal{U} \cap \mathcal{R}_R \) into \( \mathcal{U} \cap \mathcal{R}_R \) has also a least fixed point in \( \mathcal{U} \) (cf. [11]). As the functional \( E[t_1] \) maps unique relations into unique relations, it has a fixed point \( Q \) in \( \mathcal{U} \), too. \( Q \) is a unique relation and \( \mu_{E[t_1]} \subseteq Q \).

Thus, \( \mu_{E[t_1]} \) and the semantics of the deterministic functional program \( (\text{def } r \Leftarrow t_1 | t_2) \) are unique relations.

Analogously to Theorem 3.5, one can show that the mapping \( E[t] \) is continuous, too, i.e.,

\[
E[t]\left(\sup_{j \geq 1} X_j\right) = \sup_{j \geq 1} E[t](X_j)
\]
holds for every ascending chain \( X_1 \subseteq X_2 \subseteq \cdots \) of unique relations. Therefore, the least fixed point of this mapping can be represented as the least upper bound of the chain
\[
0 \subseteq E[t](0) \subseteq (E[t])^2(0) \subseteq \cdots
\]
(cf. [18]). Thus, we can use the induction principle of Scott (cf. [9, 10]) to prove properties of the least fixed point as long as they are admissible.

### 3.3. Applications

The purpose of this section is to show how to work with programs in our framework. First, we consider some of the algebraic laws in [1]. Williams [19] regards these laws as axioms. As we have defined a denotational semantics, these rules can be proved within the relational calculus. The proofs are straightforward, because we do not have to distinguish between several cases using relational algebra.

**Theorem 3.7.** Let \( t_j, 1 \leq j \leq 4 \), be functional forms, \( p_i \) a projection function symbol, and \( X \) a unique relation.

(a) \( E[[t_1, t_2] \circ t_3](X) = E[[t_1 \circ t_3, t_2 \circ t_3]](X) \); 
(b) \( E[p_1 \circ [t_1, t_2]](X) = E[t_1](X) \land E[t_2](X) \land \text{if } E[p_1](X) = \pi_1, \text{ where } (\pi_1, \pi_2) \) is the direct product corresponding to the construction; 
(c) \( E[t_1 \circ (t_2 \circ t_3; t_4)](X) = E[t_2 \circ (t_1 \circ t_3); (t_1 \circ t_4)](X) \); 
(d) \( E[[t_1, t_2 \rightarrow t_3; t_4]](X) = E[t_2 \rightarrow [t_1, t_3]; [t_1, t_4]](X) \).

**Proof.** We use the abbreviations \( E_j = E[t_j](X), 1 \leq j \leq 4 \).

(a) \[
E[[t_1, t_2] \circ t_3](X) = E_3 E[[t_1, t_2]](X)
= E_3 (E_1 \pi_1^T \land E_2 \pi_2^T)
= E_3 E_1 \pi_1^T \land E_3 E_2 \pi_2^T \quad \text{(as } E_3 \text{ is unique)}
= E[[t_1 \circ t_3, t_2 \circ t_3]](X).
\]

(b) \[
E[p_1 \circ [t_1, t_2]](X) = (E_1 \pi_1^T \land E_2 \pi_2^T) \pi_1 = E_1 \land E_2 L
\]
(cf. Section 2.1.1 and Definition 2.2(4)).

(c) \[
E[t_1 \circ (t_2 \rightarrow t_3; t_4)](X) = ((E_2 TL \land E_3) \lor (E_2 FL \land E_4)) E_1
= (E_2 TL \land E_3 E_1) \lor (E_2 FL \land E_4 E_1) \quad \text{(Theorem 2.1)}
= E[t_2 \rightarrow (t_1 \circ t_3; (t_1 \circ t_4)](X).
\]

(d) \[
E[[t_1, t_2 \rightarrow t_3; t_4]](X) = E_1 \pi_1^T \land ((E_2 TL \land E_3) \lor (E_2 FL \land E_4)) \pi_2^T
= (E_1 \pi_1^T \land (E_2 TL \land E_3) \pi_2^T) \lor (E_1 \pi_1^T \land (E_2 FL \land E_4) \pi_2^T)
= (E_2 TL \land (E_1 \pi_1^T \land E_3 \pi_2^T)) \lor (E_2 FL \land (E_1 \pi_1^T \land E_4 \pi_2^T))
= E[t_2 \rightarrow [t_1, t_3; [t_1, t_4]](X). \]
Note that the vector $RL$ characterizes the domain of a relation $R$. Therefore, $E[p_1\circ t_1, t_2](X) = E[t_1](X)$ holds if $t_2$ terminates (i.e., $E[t_2](X)$ is total).

In Theorem 3.7 we have shown that syntactically different functional forms may have the same semantics. Thus, we have actually proved some transformation rules for parts of programs. Now, we turn to a more complicated example, viz. the well-known Cooper-rule for rebracketing.

As associativity is used in this rule, we have to define this property within our framework. Let $(\pi_1, \pi_2)$ be a direct product. We call a relation $\varphi$ associative (w.r.t. $(\pi_1, \pi_2)$) if the conjunction $\pi_1 \land \pi_2 \land \varphi$ exists (or equivalently, if $\pi_1 \in R_\varphi$ and $\pi_2 \in R_\varphi$) and if

$$[Q, [R, S]_{\varphi}]_{\varphi} = [[Q, R]_{\varphi}, S]_{\varphi}$$

for arbitrary relations $Q$, $R$ and $S$. Here, $[,]$ means tupeling of relations w.r.t. the direct product $(\pi_1, \pi_2)$. If $\varphi$ is a mapping, this corresponds to the usual definition of associativity.

We want to prove that the following two programs (program schemes) have the same semantics:

$$\text{(def } r = b \rightarrow \varphi \circ [k, l]; m | \varphi \circ [p_1, p_2]),$$
$$\text{(def } s = b \circ p_1 \rightarrow s \circ [k \circ p_1, \varphi \circ [l \circ p_1, p_2]]; \varphi \circ [m \circ p_1, p_2] | s).$$

If $(\pi_1, \pi_2)$ is the direct product such that $E[p_1](X) = \pi_1$ and $E[p_2](X) = \pi_2$ and if $[,]$ is used for tupeling of relations w.r.t. $(\pi_1, \pi_2)$, then

$$\tau(Y) = (bTL \land [kY, l]\varphi) \lor (bFL \land m),$$
$$\sigma(Z) = (\pi_1 bTL \land [\pi_1, k, [\pi_1, l, \pi_2]_{\varphi}]Z) \lor (\pi_1 bFL \land [\pi_1, m, \pi_2]_{\varphi})$$

are the semantic functionals corresponding to the functional forms of the definitions above. Note that we have used the abbreviation $f$ for $i(f)$ if $f$ is a function symbol. Now, we consider the least fixed points of these functionals and prove the following theorem.

**Theorem 3.8.** Let $g$ and $h$ be the least fixed points of $\tau$ and $\sigma$, respectively. If $\varphi$ is associative w.r.t. the direct product $(\pi_1, \pi_2)$, then $[\pi_1 g, \pi_2]_{\varphi} = h$.

**Proof.** By induction, using the predicate $L(Y, Z) = ([\pi_1 Y, \pi_2]_{\varphi} = Z)$. The induction basis is trivial. In the induction step, associativity of $\varphi$ and the induction hypothesis are used:

$$[[\pi_1 k Y, \pi_1 l]_{\varphi}, \pi_2]_{\varphi} = [[\pi_1 k, \pi_1 l, \pi_2]_{\varphi}]_{\varphi} \quad \text{(as } \varphi \text{ is associative)}$$
$$= [[\pi_1 k, [\pi_1 l, \pi_2]_{\varphi}]_{\pi_1 Y, \pi_2]_{\varphi}} \quad \text{(Theorem 2.7)}$$
$$= [[\pi_1 k, [\pi_1 l, \pi_2]_{\varphi}]Z] \quad \text{(induction hypothesis)}.$$

Now the rest of the proof is rather simple.
\[ \pi_1 \tau(Y), \pi_2 ] \varphi \\
= [\pi_1 (bTL \land [kY, l]) \varphi \lor \pi_1 (bFL \land m), \pi_2 ] \varphi \\
= [\pi_1 bTL \land [\pi_1 kY, \pi_1 l], \pi_2 ] \varphi \\
\lor [\pi_1 bFL \land \pi_1 m, \pi_2 ] \varphi \quad \text{(as } \pi_1 \text{ is unique)} \\
= (\pi_1 bTL \land [(\pi_1 kY, \pi_1 l), \pi_2 ] \varphi) \\
\lor (\pi_1 bFL \land [\pi_1 m, \pi_2 ] \varphi) \quad \text{(Theorem 2.1)} \\
= (\pi_1 bTL \land [\pi_1 k, [\pi_1 l, \pi_2 ] \varphi] Z) \\
\lor (\pi_1 bFL \land [\pi_1 m, \pi_2 ] \varphi) \quad \text{(induction hypothesis and associativity)} \\
= \sigma(Z). \quad \Box \\

Theorem 3.8 yields \\
\[ M[\text{def } r = \cdots | \varphi = r \circ p_1, p_2 ] E[\varphi \circ [r \circ p_1, p_2 ] ](g) \\
= [\pi_1 g, \pi_2 ] \varphi = h = E[s](h) \\
= M[\text{def } s = \cdots | s], \]
i.e., the semantics of both programs coincide.

In this section we have dealt with program schemes, i.e., programs in which the interpretation of the function symbols \( f \in F \) is not specified. Such a scheme may be thought of as representing a family of 'concrete' programs with given interpretation. Applications of the relational calculus to concrete programs may be found in Section 5. More transformation rules are proved in [3].

4. The nondeterministic functional language NFP

In Section 3 we introduced the (deterministic) functional language DFP. The semantics of a program from DFP is a unique relation. Now we extend our language to be nondeterministic by adding the nondeterministic branching \( \Box \).

4.1. Syntax

Again we start with some given sets \( F \) and \( R \) as in Section 3.1. Then the syntax of the \textit{nondeterministic functional forms} \( \mathcal{F}_N \) over \( F \) and \( R \) is given by an extension of Definition 3.1: in addition to the rules (a) through (f) (where functional form is replaced by nondeterministic functional form) we demand

(g) if \( t_1, t_2 \) are nondeterministic functional forms, then the \textit{nondeterministic branching} \( (t_1 \Box t_2) \) is a nondeterministic functional form, too.

Given the set \( \mathcal{F}_N \), analogously to Section 3.1, one can define the set of \textit{nondeterministic} (functional) \textit{definitions} \( \text{def } r = t \) and the set \( \mathcal{P}_N \) of \textit{nondeterministic functional programs} \( \text{def } r = t_1 | t_2 ). \)
4.2. Semantics

From the literature three kinds of nondeterminism are known: angelic, demonic, and erratic nondeterminism. In angelic nondeterminism, termination is guaranteed as soon as termination is possible. This kind of nondeterminism is used in automata theory and in [12] (McCarthy's ambiguity operator). From Dijkstra's wp-calculus, demonic nondeterminism is known. Here, possible nontermination implies certain nontermination. The third kind of nondeterminism appears in Plotkin's powerdomains where defined results and undefined results may occur simultaneously.

Let \( \mathcal{R} \) be a relational algebra. To give a semantics for NFP-programs we also use an interpretation \( i : F \to \mathcal{U} \) which associates a unique relation to each function symbol.

In relational algebra it is very easy to extend the Definitions 3.4 and 3.6 such that (a first kind of) an angelic semantics for NFP-programs is obtained. Analogously to Section 3.2, the semantic functional \( E_N \) assigns a mapping \( E_N[t] : \mathcal{R} \to \mathcal{R} \) to each \( t \in \mathcal{F}_N \). \( E_N[t] \) is given by an appropriate adaptation of Definition 3.4 and the additional equation

\[
E_N[t_1 \sqcap t_2](X) = E_N[t_1](X) \lor E_N[t_2](X).
\]

Similarly to Section 3.2, \( E_N[t] \) is monotonic. Thus, its least fixed point does exist and we can define a fixed-point semantics of programs by a definition analogous to Definition 3.6.

In the same way, a demonic semantics for NFP-programs could be given. As we already mentioned, in erratic nondeterminism the interpretation of a form \( t \in \mathcal{F}_N \) may lead to the situation in which a set of defined values together with 'undefined' is a possible result. This situation cannot be described by the construction used above.

One way to describe the simultaneous occurrence of defined and undefined values is to use a special object, denoted by \( \bot \). This leads to extended relations. The ordering of relations must be changed, too. Instead of the inclusion ordering, the Egli–Milner-ordering is used. For details we refer to [7]. However, this approach has a serious drawback. A component notation of relations is needed, as \( \bot \) is an element of the extended carrier sets. Thus, the advantage of the relational calculus would be given away.

Therefore, we do not use the explicit element \( \bot \), but prefer to introduce pairs \((B, d)\) of a relation \( B \) and a vector \( d \). \( B \) can be understood as the 'breadth' of a form (i.e., the set of possible defined results) and \( d \) as its 'definedness' (i.e., the predicate that holds if and only if all possible evaluations lead to defined values). This approach is taken from [4] where the formalism of 'breadth' and 'definedness' is used to describe the applicative kernel of the programming language CIP-L. On such pairs \((B, d)\) we also want to define an ordering. We start from the well-known definition of the Egli–Milner-ordering on powerdomains:

\[
X \leq_{EM} Y \iff (\bot \notin X \text{ and } X = Y) \text{ or } (\bot \in X \text{ and } X \setminus \{\bot\} \subseteq Y).
\]

In a powerdomain the set \( X \setminus \{\bot\} \) corresponds to the breadth of a form and the
predicate \( \perp \in X \) to its definedness. Hence, we choose the equivalent definition:

\[
X \leq_{\text{EM}} Y :\iff (X \setminus \{ \perp \} \subseteq Y \setminus \{ \perp \}) \quad \text{and} \quad (\perp \notin X \implies \perp \notin Y)
\]

and \( (\perp \notin X \implies Y \setminus \{ \perp \} \subseteq X \setminus \{ \perp \}) \).

This condition can easily be expressed with the usual operations on relations. Thus, the definition of the Egli-Milner-ordering can now be given in component-free notation.

**Definition 4.1.** The *Egli-Milner-relation* \( \leq_{\text{EM}} \) on the set \( \mathcal{R} \times \mathcal{V} \) (where the symbol \( \mathcal{V} \) is used for the *set of all vectors* in the relational algebra \( \mathcal{R} \)) is defined by

\[
(B, d) \leq_{\text{EM}} (B', d') :\iff B \subseteq B' \quad \text{and} \quad d \subseteq d' \quad \text{and} \quad d \wedge B' \subseteq B.
\]

From now on we will use "\( \preceq \)" as an abbreviation for "\( \leq_{\text{EM}} \)".

Note that any chain w.r.t. \( \preceq \) is also a chain w.r.t. the component-wise inclusion ordering. In Theorem 4.2 we shall see that, for such a chain, the least upper bounds w.r.t. any of both orderings are the same. Of course, the opposite direction of this assertion is not true. E.g., \((I, L)\) is less than \((L, L)\) w.r.t. the component-wise inclusion ordering, but \((I, L) \preceq (L, L)\) does not hold (in domains with more than one element).

In contrast to the deterministic case, it is not obvious whether \((\mathcal{R} \times (\mathcal{V} \cap \mathcal{R})), \preceq)\) is a cpo (for an arbitrary relation \( \mathcal{R} \)) because \( \preceq \) differs from the component-wise inclusion ordering. This result, however, is essential if we want to apply a fixed-point theorem. Therefore, it is proved in the following theorem.

**Theorem 4.2.** \( \preceq \) is an ordering on \( \mathcal{R} \times \mathcal{V} \) and \((\mathcal{R} \times (\mathcal{V} \cap \mathcal{R})), \preceq)\) is a complete partial ordering for every relation \( \mathcal{R} \).

**Proof.** First, we have to show that \( \preceq \) is a (reflexive) ordering relation. Reflexivity and antisymmetry are obvious. To show transitivity we assume \((B, d) \preceq (B', d')\) and \((B', d') \preceq (B'', d'')\). \(B \subseteq B''\) and \(d \subseteq d''\) follow immediately. For the third condition we obtain

\[
d \wedge B'' = d \wedge d' \wedge B'' \quad \text{(as} \quad d \subseteq d')
\]

\[
\subseteq d \wedge B' \subseteq B.
\]

Clearly, \((0, 0)\) is the least element of \( \mathcal{R} \times (\mathcal{V} \cap \mathcal{R}) \) w.r.t. \( \preceq \). Let \((B_1, d_1) \preceq (B_2, d_2) \preceq \cdots \) be an ascending chain. We define

\[
lub_{j \geq 1} (B_j, d_j) = \left( \sup_{j \geq 1} B_j, \sup_{j \geq 1} d_j \right)
\]

and show that \( \text{lub}_{j \geq 1} (B_j, d_j) \) is the least upper bound of this chain w.r.t. \( \leq \). Let \( B = \sup_{j \geq 1} B_j \) and \( d = \sup_{j \geq 1} d_j \).
(a) \((B, d)\) is an upper bound: Let \((B_k, d_k)\) be an arbitrary element of the chain. \(B_k \subseteq B\) and \(d_k \subseteq d\) are trivial. For the last condition \(d_k \wedge B \subseteq B_k\), we show
\[
d_k \wedge B = \sup_{j=1} (d_k \wedge B_j) \subset B_k \vee \sup_{j=k+1} (d_k \wedge B_j),
\]
because \(B_j \subseteq B_k\) holds if \(j \leq k\). Finally, \(d_k \wedge B_j \subseteq B_k\), \(j \geq k + 1\), is obvious because of \((B_k, d_k) \leq (B_j, d_j)\).

(b) \((B, d)\) is the least upper bound: Let \((B', d')\) be an arbitrary upper bound. Then \(B \subseteq B'\) and \(d \subseteq d'\) certainly hold. As \((B_j, d_j) \leq (B', d')\) for every \(j \geq 1\), we get \(d \vee B' = \sup_{j=1} (d \vee B_j) \subset \sup_{j=1} B_j = B\) and thus, \((B, d) \leq (B', d')\). \(\square\)

In the sequel we describe an erratic semantics for the nondeterministic functional language NFP. We proceed as in Section 3.2 and define the semantics of nondeterministic functional forms by induction.

**Definition 4.3.** The erratic nondeterministic semantic functional \(E_e : \mathcal{F} \to (\mathcal{R} \times \mathcal{V} \to \mathcal{R} \times \mathcal{V})\) is inductively defined as follows:

(a) \(E_e[f](B, d) = (i(f), i(f) L)\);

(b) \(E_e[r](B, d) = (B, d)\);

(c) \(E_e[p](B, d) = (\pi_j, L)\), where \(\pi_j\) is a projection relation from an appropriate direct product \((\pi_1, \ldots, \pi_k)\), \(k \geq j\);

(d) \(E_e[[t_1 \rightarrow t_2; t_3]\](B, d) = ((B_1 T L \wedge B_2) \vee (B_1 F L \wedge B_3), d_1 \wedge (B_1 T L \vee d_2) \wedge (B_1 F L \vee d_3))\), where \((B_j, d_j) = E_e[t_j](B, d)\) for \(1 \leq j \leq 3\);

(e) \(E_e[[t_1, \ldots, t_k]\](B, d) = (\bigwedge_{j=1}^k B_j \pi_j^T \wedge \bigvee_{j=1}^k d_j)\), where \((B_j, d_j) = E_e[t_j](B, d)\) for \(1 \leq j \leq k\), and where \((\pi_1, \ldots, \pi_k)\) is a direct product such that the conjunction exists;

(f) \(E_e[[t_1 \circ t_2]\](B, d) = (B_2 B_1, d_2 \wedge B_2 d_1)\), where \((B_j, d_j) = E_e[t_j](B, d)\) for \(j = 1, 2\);

(g) \(E_e[[t_1 \sqcup t_2]\](B, d) = (B_1 \vee B_2, d_1 \wedge d_2)\), where \((B_j, d_j) = E_e[t_j](B, d)\) for \(j = 1, 2\).

We want to exclude the case that a nondeterministic functional form is 'defined' and its 'breadth' is empty, i.e., no defined result is possible. This is some sort of consistency condition on the relation and the vector part of an element of \(\mathcal{R} \times \mathcal{V}\). Of course, we expect that the functional \(E_e[t]\) preserves this condition for every nondeterministic functional form \(t\).

**Lemma 4.4.** Let \(t\) be a nondeterministic functional form and \((B, d) \in \mathcal{R} \times \mathcal{V}\) such that \(d = d L \subseteq B L\). Then \(d' = d' L \subseteq B' L\), where \((B', d') = E_e[t](B, d)\).

**Proof.** This lemma is proved by induction on the structure of \(t\). E.g., let \(t = t_1 \circ t_2\) be a composition and \((B_j, d_j) = E_e[t_j](B, d), j = 1, 2\). Then
\[
d' = d_2 \wedge B_2 \bar{d}_1 \subseteq B_2 L \wedge B_2 \bar{B}_1 L \quad \text{(induction hypothesis)}
\]
\[
\subseteq (B_2 \wedge B_2 \bar{B}_1 L L)(L \wedge B_2 \bar{B}_1 L) \quad \text{(Dedekind rule)}
\]
\[
\subseteq B_2 B_1 L = B' L \quad \text{(Schröder rule).} \quad \square
\]
Proceeding as in Section 3.2, we again prove the monotonicity of $E_c[t]$ by induction.

**Theorem 4.5.** For every $t \in T_N$ the mapping $E_c[t]: \mathbb{R} \times \mathcal{V} \rightarrow \mathbb{R} \times \mathcal{V}$ is monotonic w.r.t. $\leq$.

**Proof.** The induction basis—cases (a), (b) and (c) in Definition 4.3—is trivial.

(d) Let $t = t_0 = t_1 \rightarrow t_2; t_3$ and $(X, x) \leq (Y, y)$. We use the following abbreviations:

$$(B_j, d_j) = E_c[t_j](X, x), \quad 0 \leq j \leq 3,$$

$$(B'_j, d'_j) = E_c[t_j](Y, y), \quad 0 \leq j \leq 3.$$

$B_0 \leq B_0'$ is obvious. $d_0 \leq d_0'$ follows from $d_0 \leq d_1 \leq d_1'$ and $d_0 \leq d_1 \wedge (B_1TL \vee d_2) \leq (d_1 \wedge B_1TL) \vee d_2 = (d_1 \wedge B_1)TL \vee d_2 \leq B'_1TL \vee d'_2$ and the corresponding inclusion with $B_1FL$ and $d_3$.

The third condition for $(B_0, d_0) \leq (B_0', d_0')$ is proved as follows:

$$d_0 \wedge B_0' = (d_0 \wedge B_1TL \wedge B_1') \vee (d_0 \wedge B_1FL \wedge B_1')$$

$$= (d_1 \wedge B'_1TL \wedge (B'_1TL \wedge d_2) \wedge B_2') \vee \cdots \quad (\text{as } d_0 \leq d_1 \wedge (B_1TL \wedge d_2))$$

$$= (B_1TL \wedge d_2 \wedge B_2') \vee \cdots$$

$$= (B_1TL \wedge B_2') \vee (B_1FL \wedge B_2) \quad (\text{as } d_2 \wedge B_2' \leq B_2)$$

$$= B_0.$$

The cases (e) and (g) are easier, whereas in case (f) monotonicity is not obvious because of the negations in the definedness part. If $t = t_0 = t_1 \circ t_2$, then only the condition $B_0 \leq B_0'$ is trivially fulfilled (we use the same abbreviations as above).

Now we show the other two conditions.

$$d_0 = d_2 \wedge B_2d_1$$

$$= d_2 \wedge \overline{d_2L} \vee B_2d_1 \quad (\text{as } d_2 = \overline{d_2L})$$

$$= d_2 \wedge \overline{d_2d_1} \vee B_2d_1$$

$$= d_2 \wedge (\overline{d_2} \vee B_2) \overline{d_1}$$

$$= d'_2 \wedge B'_2d'_1 = d'_0 \quad (\text{as } d_2 \wedge B_2 \leq B_2').$$

$d_0 \wedge B_0' = d_2L \wedge B_2d_1 \wedge B_1B_1'$

$$= (d_2L \wedge B_2')B'_1 \wedge B_2d_1 \quad (\text{Theorem 2.1})$$

$$\leq B_2B'_1 \wedge B_2d_1 \quad (\text{induction hypothesis})$$

$$\leq (B_2 \wedge B_2d_1)B'_1T(B'_1 \wedge B_2B_2d_1) \quad (\text{Dedekind rule})$$

$$\leq B_2(B'_1 \wedge d_1) \quad (\text{Schröder rule})$$

$$\leq B_2B_1 = B_0 \quad (\text{induction hypothesis}) \quad \Box$$

Now it is possible to define the semantics of a nondeterministic functional program as the least fixed point of a monotonic mapping.
Definition 4.6. Let \( \text{def } r = t_1 \mid t_2 \) be a nondeterministic functional program. The \textit{erratic semantics} of this program is defined to be \( E_e[t_2](B, d) \), where \( (B, d) \) is the least fixed point w.r.t. \( \leq \) of the mapping \( E_e[t_1] \): 
\[
M_e[\text{def } r = t_1 \mid t_2] = E_e[t_2](\mu_{E_e[t_1]}).
\]

As NFP is a rather restricted language (in particular, the breadths of the basis functions are unique relations and only one definition is allowed), the functional \( E_e[t] \) is also continuous. Therefore, its least fixed point can be represented as the least upper bound of the chain of approximations
\[
(0, 0) \leq E_e[t](0, 0) \leq (E_e[t])^2(0, 0) \leq \cdots
\]
and computational induction can be used in proofs of program properties. Whereas \( E_e[t] \) is also monotonic in less restricted versions of NFP, continuity does not hold in general.

4.3. Angelic and demonic semantics

Now we change our definition of erratic semantics a little bit to get an angelic and a demonic semantics, respectively. We retain the domain \( \mathcal{R} \times \mathcal{V} \) and the ordering on this domain (the Egli–Milner-ordering, cf. Definition 4.1), too. Only the interpretation of the functional forms is changed. Thus, the angelic and the demonic semantics can also be obtained by using breadth and definedness.

Definition 4.7. The \textit{angelic nondeterministic semantic functional} \( E_a : \mathcal{S}_N \rightarrow (\mathcal{R} \times \mathcal{V} \rightarrow \mathcal{R} \times \mathcal{V}) \) and the \textit{demonic nondeterministic semantic functional} \( E_d : \mathcal{S}_N \rightarrow (\mathcal{R} \times \mathcal{V} \rightarrow \mathcal{R} \times \mathcal{V}) \) are defined like the erratic nondeterministic semantic functional \( E_e \). Their definitions differ from Definition 4.3 in the following three cases:

\textit{angelic semantics}:

\( a) \quad E_a[t_1 \rightarrow t_2; t_3](B, d) = ((B_1 \downarrow t_1) \land B_2) \lor (B_1 \downarrow t_2) \land (B_3) \lor (d_1 \lor d_2) \lor (B_3) \land (d_3) \lor (d_2 \lor d_3)); \\
\( f) \quad E_a[t_1 \circ t_2](B, d) = (B_1 \land B_2, d_1); \\
\( g) \quad E_a[t_1 \square t_2](B, d) = (B_1 \lor B_2, d_1 \lor d_2),
\)

where \( (B_j, d_j) = E_a[t_j](B, d) \) for \( 1 \leq j \leq 3 \).

\textit{demonic semantics}:

\( d) \quad E_d[t_1 \rightarrow t_2; t_3](B, d) = ((B_1 \downarrow t_1) \land B_2) \lor (B_1 \downarrow t_2) \land (B_3 \lor d_2) \land (d_3) \lor (B_3 \lor d_3) \land (d_2 \lor d_3)); \\
\( f) \quad E_d[t_1 \circ t_2](B, d) = (B_2 \land B_1, d_2 \land d_1); \\
\( g) \quad E_d[t_1 \square t_2](B, d) = ((B_1 \lor B_2) \land d_1 \lor d_2, d_1 \land d_2),
\)

where \( (B_j, d_j) = E_d[t_j](B, d) \) for \( 1 \leq j \leq 3 \).
Obviously, the erratic and the angelic semantics of nondeterministic functional forms agree in their breadth parts and differ in the definedness parts whereas the erratic and the demonic semantics have the same definedness parts but different breadth parts. This can be illustrated by the following simple example.

Let \( f \) and \( g \) be two function symbols such that \( i(f) = TT \) and \( i(g) = LF \). Then, the interpretation of \( f \) is a (partial) function which maps the truth value \( \text{true} \) to \( \text{true} \) and is undefined for the argument \( \text{false} \). The interpretation of \( g \) is the constant function yielding \( \text{false} \) for every argument. Now we consider the three kinds of nondeterministic semantics of the functional form \( f \circ g \). In the erratic and the angelic case the breadth of \( f \circ g \) is \( i(f) \lor i(g) = TT \lor LF = TL \lor FF \), i.e., both truth values are possible results if the argument is \( \text{true} \) and \( \text{false} \) is mapped only to itself. But the definedness of \( f \circ g \) is \( i(f) \land i(g) = TL \) for erratic nondeterminism (i.e., the result may be undefined if the argument is \( \text{false} \)) and \( i(f) \lor i(g) = L \) in the angelic case (i.e., \( f \circ g \) is defined for arbitrary arguments). If demonic semantics is used, the breadth of \( f \circ g \) is \( (i(f) \lor i(g)) \land i(f) \land i(g) = (TL \lor FF) \land TL = TL \) and no defined result is possible for the argument \( \text{false} \).

In angelic nondeterminism it is not possible that a functional form \( t \) would not terminate (be undefined) if it can also yield a defined result. Therefore, we only consider pairs \( (B, d) \in \mathcal{R} \times \mathcal{V} \) such that \( d = BL \). Thus, the definedness part \( d \) of \( (B, d) \) can be represented by the vector \( BL \). That is the reason why we were able to give a simpler angelic semantics at the beginning of Section 4.2. In the demonic case, possible nontermination implies that no defined result may occur, i.e., \( d \subset BL \).

Together with the consistency condition from Section 4.2, we want \( d = BL \) to hold also for demonic nondeterminism.

As in Lemma 4.4, this stronger condition has to be preserved by the angelic functional \( E_a[t] \) and by the demonic functional \( E_d[t] \).

**Lemma 4.8.** Let \( t \) be a nondeterministic functional form and \( (B, d) \in \mathcal{R} \times \mathcal{V} \) such that \( d = BL \). Then

\[
d' = d' L = B'L \quad \text{where} \quad (B', d') = E_a[t](B, d),
\]

\[
d'' = d'' L = B''L \quad \text{where} \quad (B'', d'') = E_d[t](B, d).
\]

**Proof.** The proof is quite similar to the proof of Lemma 4.4. First we show the equality for the angelic semantics by induction:

(a) and (c) are clear.

(b) was required in the assumptions.

(d) Let \( t = t_1 \rightarrow t_2 \); \( t_3 \) and \( (B_j, d_j) = E_a[t_j](B, d), 1 \leq j \leq 3 \).

\[
d' = d_1 \land (B_1 FL \lor d_2) \land (B_1 TL \lor d_3) \land (d_2 \lor d_3)
\]

\[
= (B_1 TL \lor B_1 FL) \land (B_1 FL \lor B_2 L)
\]
\[\wedge (B_1 TL \lor B_3 L) \wedge (B_2 L \lor B_2 L) \quad (\text{as } d_1 = B_1 L = B_1 TL \lor B_1 FL)\]

\[= (B_1 TL \lor B_2 L) \lor (B_1 FL \land B_3 L) \quad \text{(distributivity)}\]

\[= B'L \quad \text{(Theorem 2.1)}.\]

Cases (e), (f) and (g) are rather simple.

In the demonic case we also use induction. The cases (a), (b), (c), and (e) are the same as in angelic nondeterminism; case (g) is quite similar to case (g) before. If \(t = t_1 \circ t_2\) is a composition and \((B_j, d_j) = E_a[t_j](B, d), j = 1, 2\), we get

\[d'' = B_2 L \land \overline{B_2 d_1} \quad \text{(induction hypothesis)}\]

\[= (B_2 d_1 \lor B_2 \overline{d_1}) \land \overline{B_2 d_1}\]

\[= (B_2 d_1 \land \overline{B_2 d_1}) \lor 0 \quad \text{(distributivity)}\]

\[= B_2 B_1 L \land \overline{B_2 d_1} \quad \text{(induction hypothesis)}\]

\[= (B_2 B_1 \land \overline{B_2 d_1}) L = B''L \quad \text{(Theorem 2.1)}\]

To prove the last case (d), we use the following equations

\[B_1 L \land B_1 TL \land B_1 FL = B_1 L \land B_1 (TL \lor FL) = 0,\]

\[B_1 L = B_1 TL \lor B_1 FL = (B_1 TL \land (B_1 FL \lor B_1 \overline{FL})) \land (B_1 FL \land (B_1 TL \lor B_1 \overline{TL}))\]

\[= (B_1 TL \land B_1 FL) \lor (B_1 TL \land B_1 FL) \lor (\overline{B_1 TL} \land B_1 FL)\]

and show

\[d'' = d_1 \land (B_1 TL \lor d_2) \land (B_1 FL \lor d_3)\]

\[= (B_1 L \land B_1 TL \lor B_1 FL) \lor (B_1 L \land B_2 L \land B_3 L) \lor (B_1 L \land B_1 TL \land B_2 L) \lor (B_1 L \land B_1 TL \land B_3 L) \quad \text{(distributivity)}\]

\[= ((B_1 TL \land B_1 FL) \lor (B_1 TL \land B_1 FL) \lor (B_1 FL \land B_1 \overline{TL})) \land B_2 L \land B_2 L\]

\[\lor (B_1 TL \land B_1 FL \land B_2 L) \lor (B_1 FL \land B_1 \overline{TL} \land B_3 L)\]

\[= (B_1 TL \land B_1 FL \land B_2 L \land B_3 L) \quad \text{(distributivity)}\]

\[\lor (B_1 TL \land B_1 FL \land B_2 L \land B_3 L) \quad \text{(distributivity)}\]

\[= B''L \quad \text{(Theorem 2.1)} \square\]

Now the angelic and the demonic semantics of nondeterministic functional programs can be defined as soon as we have established the monotonicity (w.r.t. the Egli–Milner-ordering) of the functionals \(E_a[t]\) and \(E_d[t]\). This can be proved quite similarly to Theorem 4.5. Note that for an arbitrary relation \(R\) the set \(\{(B, d) \in R \times (\mathcal{V} \cap R) | d = BL\}\) is a cpo. Therefore, the fixed-point theorem for cpos can be applied.
The \textit{angelic} and the \textit{demonic semantics} of the nondeterministic functional program 
\begin{align*}
\text{(def } r &= t_1 \mid t_2)\text{ are defined by}
\end{align*}
\begin{align*}
M_a[\text{def } r &= t_1 \mid t_2] &= E_a[t_2](\mu_{E_r[t_1]}), \\
M_d[\text{def } r &= t_1 \mid t_2] &= E_d[t_2](\mu_{E_r[t_1]}).
\end{align*}
As the functionals $E_a[t]$ and $E_d[t]$ are also continuous, these least fixed points can
again be represented as least upper bounds of chains of approximations. From
Lemma 4.8 we may conclude $d = B L$, where $(B, d) = M_a[\text{def } r &= t_1 \mid t_2]$ or $(B, d) = 
M_d[\text{def } r &= t_1 \mid t_2]$.

Therefore, the angelic and the demonic semantics of a nondeterministic functional
program are completely characterized by their breadth parts. In the following
examples we do not drop the definedness parts. Thus, comparison with erratic
semantics is facilitated.

4.4. Example

In a simple example the differences between erratic, angelic, and demonic seman-
tics become apparent. Let $c$ be a symbol for a constant function. Then, its interpreta-
tion is a transposed point (cf. Section 2.1.2). We denote this point by $z = zL$. 
Furthermore, let $\text{eqc}$ be a symbol for the predicate which tests whether its argument
equals the value of $c$. The interpretation $i(\text{eqc})$ is a unique and total relation for
which the following equations hold
\begin{align*}
i(\text{eqc})_{TL} &= zL \quad \text{and} \quad i(\text{eqc})_{FL} = zL.
\end{align*}
We consider the nondeterministic functional program
\begin{align*}
(C) \quad (\text{def } r &= \text{eqc} \rightarrow c \mid (c \square r) \mid r)
\end{align*}
and obtain its erratic semantics as the least fixed point of a continuous mapping:
\begin{align*}
M_c[\text{def } r &= t \mid r] &= E_c[r](\mu_{E_c[r]}) = \mu_{E_c[r]} 
\end{align*}
(\text{where } t \text{ is an abbreviation for the functional form } (\text{eqc} \rightarrow c \mid (c \square r))).\text{ This continuous mapping } E_c[r] \text{ is given by}
\begin{align*}
E_c[t](B, d) &= ((zL \land Lz^T) \lor (\overline{zL} \land (Lz^T \lor B)), L \land (\overline{zL} \lor L) \land (zL \lor (L \land d))) \\
&= ((zL \land Lz^T) \lor (\overline{zL} \land Lz^T) \lor (zL \lor B), zL \lor d) \\
&= (Lz^T \lor (\overline{zL} \land B), zL \lor d).
\end{align*}
The angelic and the demonic semantics of the program (C) are also least fixed
points and by similar calculations we obtain the angelic semantics as $\mu_{E_c[r]}$, where
\begin{align*}
E_a[t](B, d) &= (Lz^T \land (\overline{zL} \land B), L) \\
\text{and the demonic semantics as } \mu_{E_d[r]}, \text{ where}
\end{align*}
\begin{align*}
E_d[t](B, d) &= ((zL \land Lz^T) \lor (\overline{zL} \land (Lz^T \land B) \land d), zL \lor d) \\
&= ((zL \land Lz^T) \lor (\overline{zL} \land Lz^T \land d) \lor (\overline{zL} \land B), zL \lor d). 
\end{align*}
Obviously, the three functionals are different. Now we compute approximations of
the least fixed points of the (continuous) functionals above. In the erratic case we get
\[ E_{e}[t](0, 0) = (Lz^T, zL), \]
\[ (E_{e}[t])^2(0, 0) = (Lz^T \lor (zL \land Lz^T), zL \lor zL) = E_{e}[t](0, 0). \]
Therefore, the least fixed point
\[ \mu_{E_{e}[t]} = \operatorname{lub}(E_{e}[t](0, 0)) = E_{e}[t](0, 0) \]
coincides with \( E_{e}[t](0, 0) \). Thus, the erratic semantics of the program (C) is given
by the pair \((Lz^T, zL)\). This means that termination is guaranteed only for the argument
c. For other arguments the program may not terminate but if it terminates, its result
is the constant c.

In the same way, the least fixed points of the other two functionals are calculated.
We get \((Lz^T, L)\) in the angelic case and \((zz^T, zL)\) in the demonic case. Note that
not only the functionals, but also their least fixed points disagree. If we use angelic
nondeterminism, the program (C) always terminates and yields c. Using demonic
semantics, the result c is obtained provided that the argument is also c.

5. Concrete programs

In our scheme languages DFP and NFP the interpretations \( i(f) \) of function
symbols \( f \) are arbitrary (unique) relations. In order to deal with 'concrete' programs
on given domains, we have to describe these domains in terms of relational algebra.

One example has been given in Section 2.2, viz. the characterization of the domain
\( \mathcal{B} \) of the truth values by the pair \((T, F)\). In this section we consider a more complicated
structure, the natural numbers.

**Definition 5.1.** The relational system \( \mathcal{N} \) is defined as a triple \((N; z, s)\), where \( z \) and
\( s \) are relations on the set \( N \) such that \( s \) is an injective function, \( z \) is a point, and
the following properties are fulfilled:

1. \( sz = 0; \)
2. \( L \) is the least fixed point of the (monotonic and continuous) mapping \( \tau_N(X) = z \lor s^T X \), i.e., \( L = \inf\{X | \tau_N(X) \subseteq X\} \).

Clearly, the natural numbers \( \mathbb{N} \) with zero and with the successor function are a
model of this system. Definition 5.1 can be regarded as a relational variant of the
well-known Peano-axioms: (1) is the relational version of the law 'there is no natural
number with successor zero', and (2) corresponds to the induction axiom.

Note that also uniqueness, totality, and injectivity of \( s \) and bijectivity of \( z \) can
be described with relational algebraic means:

3. \( s^Ts \subseteq I, \)
5. \( zz^T \subseteq I, \)
4. \( ss^T = I, \)
6. \( z^T z \supseteq I. \)
$s^T$ is a unique (and bijective) relation which corresponds to the predecessor function on the natural numbers.

As we have used the induction axiom (2) in Definition 5.1, only finitely generated models of $\mathcal{N}$ are possible. In the following theorem, it is shown that the characterization in Definition 5.1 is even monomorphic, i.e., there exists an isomorphism between any pair of models of the system $\mathcal{N}$ (cf. [14]).

**Theorem 5.2.** Let $\mathcal{N}_i = (N_i; z_i, s_i), \; i = 1, 2$, be two systems fulfilling the laws in Definition 5.1. The least fixed point $\phi$ of the continuous functional $\sigma(X) = z_1 z_2^T \vee s_1^T X s_2$ is a bijective mapping (in the relational algebraic sense) with

$$z_1 \phi = z_1 = \phi z_2, \quad s_1 \phi = \phi s_2.$$ 

This means that $\phi$ is an isomorphism between $\mathcal{N}_1$ and $\mathcal{N}_2$.

**Proof.** Continuity of $\sigma$ is trivial. Hence, we can use computational induction.

(a) Uniqueness of $\phi$ is shown by choosing $\mathcal{L}(X) = (X^T X \in I)$. Clearly, $\mathcal{L}(0)$ holds. The induction step follows from

$$\sigma(X)^T \sigma(X) = (z_1 z_2^T \vee s_1^T X s_2)^T (z_1 z_2^T \vee s_1^T X s_2)$$

$$= z_2 L z_2^T \vee s_2^T X s_1 s_1^T X s_2 \quad (z_1 z_1 = L, s_1 z_1 = 0)$$

$$f I \vee s_2^T s_2 \quad (s_1 \text{ injective, induction hypothesis})$$

$$= I \quad (s_2 \text{ unique}).$$

(b) For totality, we prove that $\phi L$ is a fixed point of $\tau_{\mathcal{N}_1}$.

$$\phi L = \sigma(\phi) L = z_1 z_2^T L \vee s_1^T \phi s_2 L = z_1 L \vee s_1^T \phi L = \tau_{\mathcal{N}_1}(\phi L),$$

since $z_2$ is surjective and $s_2$ total. Property (2) yields $L \subseteq \phi L$.

(c) Injectivity of $\phi$ is shown analogously to (a); surjectivity analogously to (b).

(d) Only the two equations remain to be shown. As $z_1 L = z_1$, $s_2 z_2 = 0$, and $\phi = \sigma(\phi)$, we have

$$\phi z_2 = (z_1 z_2^T \vee s_1^T \phi s_2) z_2 = z_1 z_2 z_2 = z_1 L = z_1 L \phi = z_1 \phi.$$ 

For the second equation, we use $s_1 z_1 = 0$, $\phi = \sigma(\phi)$, and $s_1 s_1^T = I$ (cf. (4)):

$$s_1 \phi = s_1 (z_1 z_2^T \vee s_1^T \phi s_2) = s_1 s_1^T \phi s_2 = \phi s_2.$$ 

Note that the uniqueness of $\phi$ (and the injectivity of $\phi$, i.e., the uniqueness of the transposed relation $\phi^T$) can be proved without using property (2). Only the proofs of totality and surjectivity require (2). This fact is illustrated by viewing $\sigma$ as the semantic functional of a functional form $t$. Totality means that the program $(\mathsf{def} \; r = t \mid r)$ has to terminate, and termination requires that the domains are finitely generated.
Therefore, if the generation-principle is described by a functional as in (2), it is possible to give proofs for totality of relations and thus proofs for ‘termination of programs’ by Scott’s induction principle. In the usual theory of semantics of applicative programs (cf. [9, 10]), such proofs are impossible or rather complicated.

We consider the following concrete DFP-program over natural numbers

(A) \[ \text{(def } r = (\text{eq0} \circ p_1) \rightarrow p_2 ; (\text{succ} \circ r \circ [\text{pred} \circ p_1, p_2]) | r) . \]

Let \((N; z, s)\) be as in Definition 5.1. Then, the interpretation of the function symbols in (A) is given as follows: \(i(\text{eq0})\) is a unique and total relation such that \(i(\text{eq0})T = zT\) and \(i(\text{eq0})F = zF\) (cf. Section 4.4), \(i(\text{succ}) = s\), and \(i(\text{pred}) = s^T\). Furthermore, let \((\pi_1, \pi_2)\) be a direct product corresponding to the projection function symbols \(p_1\) and \(p_2\). It is easy to see that this program describes the addition of natural numbers.

We want to prove that the program (A) always terminates. The semantics of (A) is the least fixed point of the (continuous) functional

\[
\tau(X) = E[(\text{eq0} \circ p_1) \rightarrow p_2 ; (\text{succ} \circ r \circ [\text{pred} \circ p_1, p_2])](X)
\]

\[= (\pi_1zT \land \pi_2) \lor (\pi_1zT \land [\pi_1s^T, \pi_2]Xs)
\]

\[= (\pi_1zT \land \pi_2) \lor [\pi_1s^T, \pi_2]Xs,
\]

where \([., .]\) denotes tupeling of relations w.r.t. the direct product \((\pi_1, \pi_2)\). The last equation holds because, from Definition 5.1(1), we obtain by Schröder’s rule \(s^T \subset z\) and thus,

\[ [\pi_1s^T, \pi_2]Xs \subset \pi_1s^T \pi_2Xs \subset \pi_1s^T L \subset \pi_1zT \]

We prove the totality of the least fixed point of \(\tau\) using computational induction.

**Theorem 5.3.** Let \(g\) be the least fixed point of the mapping \(\tau\). Then \(gL = g\).

**Proof.** We choose \(L(Y, Z) = \text{(eq0} \land \pi_1ZL\) as an admissible predicate. \(L(0, 0)\) holds. If \(L(Y, Z)\) is fulfilled, then we have \(L(\tau(Y), \tau_N(Z))\), because

\[
\tau(Y)L = (\pi_1zT \land \pi_2)L \lor [\pi_1s^T, \pi_2]YsL
\]

\[= \pi_1zT \lor [\pi_1s^T, \pi_2]YL \quad (s \land \pi_2 \text{ total, Theorem 2.1})
\]

\[= \pi_1zT \lor [\pi_1s^T, \pi_2]\pi_1ZL \quad (\pi_1 \text{ unique, cf. Section 2.1.1})
\]

\[= \pi_1zT \lor \pi_1s^T ZL \quad (\pi_2 \pi_1 = L, \pi_2 \text{ total})
\]

\[= \pi_1\tau_N(Z)L.
\]

From Definition 5.1(2), it now follows that \(gL = \pi_1L = L\), i.e., \(g\) is total. \(\square\)

In the same way, concrete NFP programs can be investigated. E.g., consider the ‘less-or-equal’ relation which associates the number \(n\) itself and all natural
numbers that are less than \( n \) with the natural number \( n \) (cf. the 'less function' in [12]). It is defined by the NFP-program

\[(\text{LE}) \quad (\text{def } r = \text{eq} 0 \to \emptyset; (\text{id } \circ (r \circ \text{pred})) \mid r),\]

where \( \emptyset \) is a function symbol for the constant function yielding zero and id is the symbol for the identity function on the natural numbers.

This program always terminates and has the same semantics w.r.t. all three kinds of nondeterminism, viz. the pair \((s^T, L)\), where \( s^T \) denotes the reflexive and transitive closure of the relation \( s^T \).

Acknowledgment

We gratefully acknowledge a number of valuable discussions with G. Schmidt and W. Dosch. We thank H. Ehler for carefully reading a draft version of this paper.

References


