Regularity of relations: A measure of uniformity*

A. Jaoua and A. Mili

Department of Informatics, Faculty of Sciences of Tunis, Belvedere 1002, Tunis, Tunisia

N. Boudriga

Department of Mathematics, Faculty of Sciences of Tunis, Belvedere 1002, Tunis, Tunisia

J. L. Durieux

Laboratoire L.S.I., Université Paul Sabatier, 31077 Toulouse, France

Communicated by M. Nivat
Received January 1988
Revised November 1988

Abstract


In their most general form, program specifications can be represented as binary relations. The study of binary relations for the purpose of discussing program construction, program fault tolerance and program exception handling have led us to discover an interesting property of relations: regularity. The interest of this property is twofold: first it is very general, i.e. it is verified by several specifications we encounter; second, it is very strong, i.e. it allows us to simplify our formal computations rather dramatically.

1. Introduction: why regular relations?

Binary relations have proved to play an important role in several aspects of structured programming: as a versatile mathematical tool for teaching programming basics and methodology [15] and for defining program correctness; as a basis for program specifications [1, 8, 11, 19, 21]; as a means for capturing program meaning and programming language semantics [6, 15, 16, 17]; as a tool for defining and recording design decisions [6, 15, 13]; as a foundation for program fault tolerance [5, 10] and program exception handling [2, 5].

* This work is partly supported by the Tunisian Commission for Scientific and Technical Research, the Ministry of Higher Education and Scientific Research, under grants number MA7/85/FST, MA1/88/FST, MA4/88/FST and MA5/88/FST.

0304-3975/91/$03.50 © 1991—Elsevier Science Publishers B.V. (North-Holland)
The study of these aspects has led us to undertake a great deal of algebraic manipulations of binary relations. Such manipulations include: the resolution of optimization problems, for the purpose of determining an optimal executable assertion [4]; the calculation of complex relational expressions, for the purpose of determining the critical information of a program under the hypothesis of non-deterministic specifications [5]; the construction of relational specifications for the purpose of characterizing error recovery routines [5]; the derivation of a while statement from a non-deterministic specification [12, 13]; and so on.

At several steps in our computations, we found that we could not make progress unless we made an assumption about the specification at hand. Specifically, we had to assume that the specification meets some property, which we call regularity.

Regular relations are the subject of this paper. In Section 2 we give some mathematical background, then define regular relations. In Section 3, we present the mathematical manipulations that have led us to introduce regular relations, and show how they are affected by this property. In Section 4, we characterise regular relations, showing in particular how general they are. In Section 5, we discuss how to construct a regular closure of a given relation (specification). Finally, in Section 6, we investigate prospects for future research.

2. Regular relations: definitions

2.1. Background

A binary relation on set $S$ is a set of pairs of $S$, i.e. a subset of $S \times S$. The operations of union and intersection of relations are denoted by (respectively) $\cup$ and $\cap$. In addition, we define the following operations on the set of relations on a given set $S$:

- the **image set** of element $s$ by relation $R$ is
  \[ s.R = \{ s' \mid (s, s') \in R \} \];
- the **inverse** of relation $R$ is the relation
  \[ R^\circ = \{ (s, s') \mid (s', s) \in R \} \];
- the **subidentity** of set $A$ is the relation
  \[ I(A) = \{ (s, s) \mid s \in A \} \];
- the **relative product** of relation $R$ by relation $R'$ is the relation
  \[ R \cdot R' = \{ (s, s') \mid \exists s'': (s, s'') \in R \land (s'', s') \in R' \} \];
- the **kernel** of relation $R$ is the relation
  \[ K(R) = \{ (s, s') \mid \emptyset \neq s'.R \subseteq s.R \}; \]
- the **nucleus** of relation $R$ is the relation $N(R) = R \cdot R^\circ$.

It is noteworthy that $K(R)$ is reflexive and transitive for any relation $R$, while $N(R)$ is reflexive and symmetric for any relation $R$. 

Also, among the non-trivial properties of relations, we define the following. Let \( R \) and \( R' \) be two relations on space \( S \); we say that \( R \) is more-deterministic than \( R' \) if and only if
\[
R^* R \supseteq R'^* R'.
\]
Intuitively speaking \( R \) is more-deterministic than \( R' \) if and only if it has a lower image per argument ratio. Relation \( R \) is said to be deterministic if and only if it is more-deterministic than the identity \( I \), i.e.
\[
R^* R \subseteq I;
\]
we then say that \( R \) is a function.

Let \( R \) and \( R' \) be two relations on space \( S \); we say that \( R \) is more-defined than \( R' \) if and only if
\[
\text{dom}(R') \subseteq \text{dom}(R), \quad \forall s \in \text{dom}(R'): \quad s.R \subseteq s.R'.
\]
It is shown [11] that this property means intuitively that \( R \) carries more input/output information than \( R' \).

2.2. Regular relation

We say that \( R \) is regular if and only if
\[
R \ast R^* R \subseteq R.
\]
It is noteworthy that for all relation \( R \), we have
\[
R \subseteq R \ast R^* R;
\]
so that regularity can be defined as
\[
R = R \ast R^* R.
\]
We will, in the future, use one formula or the other, interchangeably.

2.3. Specifying with relations

In their seminal paper on program specifications, Liskov and Berzins [7] distinguish between functional specifications and performance specifications. Within functional specifications, they distinguish between procedural aspects and data aspects. In this paper, we concentrate on procedural aspects of functional specifications.

We have shown [11] that within these confines, binary relations play a capital role. Specifically, a specification is a relation containing all admissible input/output pairs. The best way to formally define the significance of a specification is to state under what condition a program is correct with respect to this specification. We offer this definition (from [11]). Program \( p \) is said to be correct with respect to \( R \) if and only if the function computed by \( p \) is more-defined than \( R \).
As an illustration, let us give examples of specifications on some space $S = \text{real}$. The purpose of this example is to exhibit the expressive power of relations as specifications.

**Specification $R_0$** requires candidate programs to be defined for non-negative arguments $s$; the output is required to have a square between $s - 1$ and $s + 1$.

**Specification $R_1$** requires programs to be defined for non-negative arguments $s$; the output is required to have a square precisely equal to $s$. Notice that both $R_0$ and $R_1$ are non-deterministic, since for any given value of $s^2$, there exist two values of $s'$.

**Specification $R_2$** requires candidate programs to be defined over all of space $S$; $R_2$ imposes the same requirement as $R_1$ for positive or null arguments, and it imposes no requirement on the output associated with a negative argument.

**Specification $R_3$** requires candidate programs to be defined over all of space $S$; while for non-negative arguments $s$ it requires candidate programs to compute a (positive or negative) square root of $s$, for negative arguments it requires candidate programs to compute a (positive or negative) square root of $-s$. The difference in behaviour for negative arguments, between $R_1$, $R_2$, and $R_3$ is the following: $R_1$ does not care whether candidate programs are defined for negative arguments; $R_2$ does care that candidate programs be defined for negative arguments, but does not care what output is associated with them; $R_3$ requires candidate programs to be defined for negative arguments, and requires the associated output to be a square root of the argument.

**Specification $R_4$** is similar to specification $R_3$ except that for negative arguments, it requires that the output be the exact positive root of the argument.

It is left to the reader to verify that $R_0 \ldots R_4$ are ranked by order of increasing definedness (i.e., by virtue of the interpretation of the relation more-defined, that they are more and more stringent); as a consequence (and by virtue of the transitivity of the relation more-defined), any program that is correct with respect to a given relation is correct with respect to all the relations with a lower index.

3. Regular relations: their interest

We have been doing work for some time now, on various aspects of structured programming. In all these problems, we deal with program specifications under the form of binary relations: the specification from which we are deriving a correct program; the specification with respect to which we are planning error recovery; the specification with respect to which we are making the program self-checking.
In four different instances, we have encountered situations where our computations are dramatically facilitated if we take the hypothesis that the specification at hand is regular. We review these situations in turn, below.

3.1. On the design of while loops from non-deterministic specifications

The design of an iterative program of the form

\[
\text{while } t \text{ do } b
\]

from a non-deterministic specification (relation) \( R \) requires, at some point, that we exhibit in relational form the specification of the loop body \( b \), as well as a correct expression for condition \( t \). In [13], we have found that the specification can be obtained by solving the following equations in \( B \):

\[
R^+ = T, \quad \text{where } T = I(S - \text{rng}(R)) \ast K(R) \circ GT
\]

for some well founded ordering relation \( GT \). As for condition \( t \), it has the logical expression \((s \in \text{dom}(B))\).

The resolution of the equation above requires that we compute \( K(R) \) which captures, in effect, the invariant (in the sense of Hoare [3]) features of relation \( R \) [13]. Now, as is clear from its definition, \( K(R) \) is nearly impossible to compute, even for very simple relations. We have found, however, that when \( R \) is regular, \( K(R) = N(R) \).

Example 3.1. We let the space be defined by the following declarations:

\[
a, b, c: \text{ integer;}
\]

and we let the specification be

\[
R = \{(s, s') | c(s') = c(s) + a(s) \ast b(s) \& b(s') = 0\};
\]

notice that \( R \) is non-deterministic since it specifies no value for \( a(s') \). We find (consult [12, 13] for further details)

\[
K(R) = \{(s, s') | c(s) + a(s) \ast b(s) = c(s') + a(s') \ast b(s')\}.
\]

From this expression of \( K(R) \) we derive

\[
B = \{(s, s') | b(s) \neq 0 \& b(s') = b(s) - 1 \& c(s') = c(s) + a(s)\};
\]

from \( B \) we extract condition \( t \) and loop body \( b \) as:

\[
t: \quad b \neq 0
\]

\[
b: \quad \text{begin } b := b - 1; \ c := c + a \text{ end.}
\]

Whence the program

\[
\text{while } b \neq 0 \text{ do begin } b := b - 1; \ c := c + a \text{ end.}
\]
3.2. On the design of initialized while loops from non-deterministic specifications

Given a relation $R$, that represents the specification of an initialized while statement, we want to decompose it into the product of two relations, say $J$ and $W$, such that if a while statement $w$ is constructed to be correct with respect to $W$ and an initialization segment $\text{init}$ constructed to be correct with respect to $J$ then the program

\begin{verbatim}
begin init; w end
\end{verbatim}

is correct with respect to $R$. We have found \cite{12, 13} that $J$ and $W$ can be extracted from $R$ as follows:

First: find relation $W$ such that
\[
\text{dom}(K(R) \circ I(\{s: s.\ W \subseteq s.R\})) = S,
I(\text{rng}(W)) \subseteq W.
\]

Second: take
\[J = K(R) \circ I(\{s: s.\ W \subseteq s.R\}).\]

Example 3.2. If we take the same space as Example 3.1, and consider the following specification,
\[R = \{(s, s') | b(s) \geq 0 \land b(s') = 0 \land c(s') = a(s) \ast b(s)\},\]
then (do we assume without proof) the relation given in Example 3.1 is an admissible solution for $W$.

3.3. On forward error recovery

In the study of forward error recovery \cite{5}, the kernel of relation $R$, the specification of the program at hand, appears twice: first as a characterization of the critical information of the program's state, i.e. the information that must be preserved in order to ensure the survival of the program; this characterization is
\[(s_0, s) \in K(R),\]
where $s_0$ is the initial state, and $s$ is the current state of the program. Second, the kernel appears in the formula of the specification of recovery routines, which is
\[J = K(R) \circ I(D),\]
where $D$ is the domain on which the while statement is correct with respect to $R$.

Now, again, the hypothesis that $R$ is regular makes the computation of $K(R)$ a great deal easier.

Example 3.3. Let $S$ be the space defined by an array of size $a$, say $a$, and an index, say $k$; we let $R$ be the specification of a sorting program. Then the critical information of this program is captured by the relation
\[K(R) = \{(s, s') | \text{permutation}(a(s), a(s'))\},\]
and the recovery routine is defined by

\[
J = \{(s, s') | \text{permutation}(a(s), a(s')) \& \text{partially-ordered}(a(s'), 1..k(s'))\}.
\]

### 3.4. On the design of executable assertions

In the implementation of program fault tolerance, one of the key phases consists of detecting errors (the detectable manifestation of faults) and assessing damage. In [5, 4] we have formalized this problem as the resolution of an optimization problem in the calculus of binary relations. To determine an executable assertion that checks for the correctness of the current state, we must solve the following optimization problem (in the unknown \(X\)):

\[
\text{MIN} \text{ (definedness}(X))
\]

{i.e. find the least defined relation \(X\) such that}

\[
\begin{align*}
\text{dom}(X) &= \text{dom}(F \ast R^*) \\
R \ast F^* X \subseteq R \ast F^*,
\end{align*}
\]

where \(F\) is some function. If \(R\) is regular, then we have the feasible solution [4]

\[
X = F \ast R^* \ast R \ast F^*.
\]

We can prove, further, that \(X\) is optimal. Let \(X'\) be feasible; we prove that \(X'\) is more-defined than \(X\). Because both have the same domain (\(\text{dom}(F \ast R^*)\)), this amounts to proving that \(X' \subseteq X\).

When \(R\) is not regular, we cannot offer a constructive, general solution.

**Example 3.4.** We consider the following program:

\[
p = \text{begin } k := 1; f := 1; \text{while } k \neq n + 1 \text{ do begin } f := f \ast k; k := k + 1 \text{ end end}
\]

on space \(n, f, k\): integer, and we consider the specification

\[
R = \{(s, s') | n(s') = n(s) \& k(s') = n(s) + 1\}.
\]

In order to check the integrity (defined formally in [4]) of \(p\) with respect to \(R\), we must execute the following executable assertion at each iteration:

\[
n(s') = n(s) \& k(s') \leq n(s') + 1,
\]

where \(s\) and \(s'\) are program states at consecutive iterations.

### 4. Regular relations: characterizations

We have found regular relations to be very general; in particular, all but the most pathological specifications we encounter in practice are regular. In this section, we
discuss characterization of regular relations that will elucidate why they are so
general. We will discuss, in turn, sufficient conditions, necessary conditions, then
necessary and sufficient conditions of regularity.

4.1. Sufficient conditions

We have identified several sufficient conditions for regularity. Some of them are
trivial, and can be readily proven; we give them below.

If a relation is deterministic (i.e. is a function), is symmetric and transitive, or is
rectangular then it is regular. On the other hand, the inverse of a regular relation,
and the cartesian product of two regular relations, are regular. Because most spaces
we work on are cartesian products of simpler spaces, this proposition is often useful
(as often as we find relations that respect the cartesian boundaries of the space).

Other sufficient conditions of regularity warrant an individual study, either because
of their interest or because of their non-triviality.

**Proposition 4.1.** The intersection of two regular relations is regular.

**Proof.** Let \( R \) be \((R_1 \circ R_2)\).

\[
(R_1 \circ R_2) \ast (R_1 \circ R_2) \ast (R_1 \circ R_2) \subseteq R_i \ast R_i \ast R_i \quad \text{for } i = 1, 2.
\]

Hence

\[
(R_1 \circ R_2) \ast (R_1 \circ R_2) \ast (R_1 \circ R_2) \subseteq R_1 \circ R_2.
\]

**Example 4.2.** Let \( S \) be the set defined by the following variable declarations:

\[
a, b, c : \text{ integer},
\]

and let \( R \) be the relation defined by

\[
R = \{(s, s') | a(s') = a(s) + b(s) \& b(s') = 0\}.
\]

This relation can be written as \( R = R_0 \circ R_1 \), where

\[
R_0 = \{(s, s') | a(s') + b(s') = a(s) + b(s)\},
R_1 = \{(s, s') | b(s') = 0\}.
\]

\( R_0 \) is regular since it is an equivalence; relation \( R_1 \) is regular since it is rectangular;
hence so is \( R \), by virtue of Proposition 4.1.

This proposition is useful in practice, since it is common for specifications to be
written as the intersection of an equivalence relation with a rectangular relation [11].

In [11], two strategies were presented for the stepwise construction of
specifications: the intersection strategy, which constructs the target specification as
the intersection of elementary specifications; the union strategy, which constructs
the target specification as the union of elementary specifications, which meet a
pairwise property. The proposition above provides that regularity is closed with respect to the intersection strategy. The question that we wish to address, naturally, is whether regularity is also closed with respect to the union strategy.

Relations \( R_1 \) and \( R_2 \) are said to be relatively uniform if and only if

\[
\forall s, s': \ s.R_1 \cap s'.R_2 \neq \emptyset \Rightarrow s.R_1 = s'.R_2.
\]

In the union strategy [11], we propose to perform the union of elementary relations under the following condition

\[
\forall s: s \in \text{dom}(R_1) \cap \text{dom}(R_2) \Rightarrow s.R_1 = s.R_2.
\]

We would have expected that these conditions be equivalent (to mean that the union strategy preserves regularity); unfortunately, such is not the case, since the latter condition is weaker than the former. Nevertheless, for the decompositions that meet the condition of relative regularity, the proposition below has some interest.

**Proposition 4.3.** If \( R_1 \) and \( R_2 \) are regular, and relatively uniform, and \( R_1^\triangleright \), \( R_2^\triangleright \) are relatively uniform, then

\[
R = R_1 \cup R_2
\]

is regular.

**Proof.** We compute

\[
R \ast R_1^\triangleright * R = (R_1 \cup R_2) \ast (R_1 \cup R_2)^\triangleright \ast (R_1 \cup R_2)
\]

\[
= R_1 \ast R_1^\triangleright \ast R_1 \cup R_1 \ast R_1^\triangleright \ast R_2 \cup R_1 \ast R_2^\triangleright \ast R_1 \cup R_1 \ast R_2^\triangleright \ast R_2 \cup R_2 \ast R_1^\triangleright \ast R_1 \cup R_2 \ast R_1^\triangleright \ast R_2 \cup R_2 \ast R_2^\triangleright \ast R_1 \cup R_2 \ast R_2^\triangleright \ast R_2.
\]

Clearly, the first and last terms of this expression are subsets of \( R \). Due to the interchangeability of \( R_1 \) and \( R_2 \) and to the preservation of regularity by inversion, it suffices that we prove the inclusion of the following terms in \( R \):

\[
R_1 \ast R_1^\triangleright \ast R_2, \quad R_1 \ast R_2^\triangleright \ast R_1, \quad R_1 \ast R_2^\triangleright \ast R_2.
\]

**Proof of** \( R_1 \ast R_1^\triangleright \ast R_2 \subseteq R \): Let \((x, y)\) be in \( R_1 \ast R_1^\triangleright \ast R_2 \); then there exists \( u \) such that

\[
(x, u) \in R_1 \ast R_1^\triangleright \ast R_2.
\]

Because \( u.R_1^\triangleright \ast \emptyset \), and \( R_1^\triangleright \) and \( R_2^\triangleright \) are relatively uniform, \( u.R_1^\triangleright = y.R_2^\triangleright \). Hence \((y, x)\) \( \in R_2^\triangleright \); whence

\[
(x, y) \in R_2 \subseteq R.
\]

**Proof of** \( R_1 \ast R_2^\triangleright \ast R_1 \subseteq R \): Let \((x, y)\) be in \( R_1 \ast R_2^\triangleright \ast R_1 \); then there exist \( u \) and \( v \) such that

\[
(x, u) \in R_1 \ast (u, v) \in R_2^\triangleright \ast (v, y) \in R_1.
\]
Because $x.R_1 \cup v.R_2 \neq \emptyset$, and $R_1$ and $R_2$ are relatively uniform, $x.R_1 = v.R_2$. Hence $(x, y) \in R_1$; whence $(x, y) \in R$.

Proof of $R_1 \ast R_2^\ast \ast R_2 \subseteq R$: Let $(x, y)$ be in $R_1 \ast R_2^\ast \ast R_2$; then there exists $u$ and $v$ such that

$$(x, u) \in R_1 \ast (u, v) \in R_2^\ast \ast (v, y) \in R_2.$$ 

Because $x.R_1 \cup v.R_2 \neq \emptyset$, and $R_1$ and $R_2$ are relatively uniform, $x.R_1 = v.R_2$. Hence $(x, y) \in R_1$, whence $(x, y) \in R$. □

Example 4.4. Let $S$ be defined by the following declaration:

\begin{align*}
& a, b: \text{ integer}, \\
\end{align*}

and let $R_1$ and $R_2$ be the following relations:

\begin{align*}
& R_1 = \{(s, s') | a(s) \leq b(s) \& s' - s\}, \\
& R_2 = \{(s, s') | a(s) \geq b(s) \& a(s') = b(s) \& b(s') = a(s)\}.
\end{align*}

It is trivial to check that $R_1$ and $R_2$ are relatively uniform; in addition, they are regular. We can check the same thing about $R_1^\ast$ and $R_2^\ast$. Hence their union (which defines the specification of a sorting program) is regular.

Proposition 4.5. The right relative product of a regular relation by a function is regular.

Proof. Let $R$ be $f \ast R_1$, where $f$ is a function and $R_1$ a regular relation.

\begin{align*}
R \ast R^\ast \ast R &= f \ast R_1 \ast R_1^\ast \ast f \ast f \ast R_1 \ast R_1 \\
& \subseteq f \ast R_1 \ast R_1^\ast \ast R_1 \quad \text{because } f \text{ is a function} \\
& \subseteq f \ast R_1 \quad \text{because } R_1 \text{ is regular} \\
& \subseteq R \quad \text{by definition.} \quad \Box
\end{align*}

Example 4.6. Let $S$ be the space defined by the following declarations:

\begin{align*}
& a, b, c: \text{ integer}, \\
\end{align*}

and let $R$ be the relation

\begin{align*}
R &= \{(s, s') | a(s') = b(s) + a(s)\}.
\end{align*}

$R$ may be written as the relative product of the function

\begin{align*}
f &= \{(s, s') | a(s') = b(s) + a(s) \& c(s') = c(s) \& b(s') = b(s)\}
\end{align*}

and the equivalence relation

\begin{align*}
R_1 &= \{(s, s') | a(s') = a(s)\}.
\end{align*}

$R_1$ is regular because it is an equivalence. Hence, by Proposition 4.5, so is $R$. 
This proposition is useful in practice, since it is common for specifications to be written as the conjunction of several predicates that have the form \( x(s') = h(s) \), where \( x \) is a variable of the space at hand, and \( h \) is a function. If we have such a conjunct for each variable of the space, then the resulting relation is regular, because it is a function. What Proposition 4.5 provides is that we have a regular relation even when not all variable of state \( s' \) are specified.

4.2. Necessary conditions

Generally, the nucleus of relation \( R, N(R) \), is reflexive and symmetric, but is not necessarily transitive; also, the kernel of relation \( R, K(R) \), is reflexive and transitive, but is not necessarily symmetric. Regularity arranges all of this. We give below three necessary conditions of regularity, the two first without proof or illustration, for they are trivial.

**Proposition 4.7.** If \( R \) is regular then \( N(R) \) is transitive.

**Proposition 4.8.** If \( R \) is regular then \( K(R) \) is symmetric.

From Propositions 4.7 and 4.8 it stems that both \( N(R) \) and \( K(R) \) are equivalences. This is a forerunner for the next proposition.

**Proposition 4.9.** If \( R \) is regular then \( N(R) = K(R) \).

**Proof.** The identity \( K(R) \subseteq N(R) \) holds for any relation \( R \), regardless of whether it is regular. Let \( (s, s') \) be an element of \( N(R) \):

\[
\exists u: \ (s, u) \in R \land (s', u) \in R
\Rightarrow s.R \cup s'.R \neq \emptyset
\]

if we invoke Proposition 4.10, which is proven independently, we can deduce from the above that

\[
\Rightarrow s.R = s'.R \land s.R \neq \emptyset
\Rightarrow s.R \neq \emptyset \land s'.R \subseteq s.R
\Rightarrow (s, s') \in K(R).
\]

4.3. Necessary and sufficient conditions

Relation \( R \) on \( S \) is said to be **uniform** if and only if

\[
\forall u, v \in \text{dom}(R): \ u.R \cup v.R \neq \emptyset \Rightarrow u.R = v.R.
\]

A uniform relation is one whose image sets are either disjoint or identical. Their precise structure will be further elucidated in this section.
Proposition 4.10. \( R \) is regular if and only if \( R \) is uniform.

Proof. Proof of sufficiency: Let \( R \) be uniform, and let \((s, s')\) be in \( R \times R \times R \). There exists \( u, v \) such that
\[
(s, u) \in R \land (v, u) \in R \land (v, s') \in R.
\]
Because it includes \( u \), \( s.R \cup v.R \) is not empty. By virtue of uniformity, we have
\[
s.R = v.R.
\]
Because \( s' \in v.R \), it also belongs to \( s.R \). Hence \((s, s') \in R\).

Proof of necessity: Let \( R \) be regular and let \( u \) and \( v \) be two elements of its domain such that
\[
u.R \cup v.R \neq \emptyset.
\]
Let \( w \) be an element of \( u.R \cup v.R \), and let \( z \) be an element of \( u.R \).
\[
(v, w) \in R, \quad (w, u) \in R, \quad (u, z) \in R.
\]
Hence \((v, z) \in R \times R \times R \); because \( R \) is regular, \((v, z) \in R \). We have deduced \((v, z) \in R \) from \((u, z) \in R \). Hence \( u.R \subseteq v.R \). Because \( u \) and \( v \) play symmetric roles, we have \( u.R = v.R \).

Example 4.11. Let \( S \) be the set of arrays (in the Pascal sense) of integers of size \( n \) and let \( R \) be the relation:
\[
R = \{(s, s') | \text{sum}(s') = \text{sum}(s) \land \text{sorted}(s')\}.
\]
Let \( u, v \in S \) such that
\[
u.R \cup v.R \neq \emptyset.
\]
Then,
\[
u.R = \{s | (\text{sum}(s) = \text{sum}(u)) \land \text{sorted}(s)\}
\]
\[
v.R = \{s | (\text{sum}(s) = \text{sum}(v)) \land \text{sorted}(s)\}.
\]
If \( s0 \in u.R \cup v.R \) then
\[
\text{sum}(s0) = \text{sum}(u) \land \text{sorted}(s0) \land \text{sum}(s0) = \text{sum}(v) \land \text{sorted}(s0)
\]
\[
\Rightarrow \text{sum}(u) = \text{sum}(v)
\]
\[
\Rightarrow u.R = v.R.
\]
Hence, this relation is regular.

Relation \( R \) is said to be rational if and only if there exist two functions \( h \) and \( g \) such that \( R = h \circ g \). Note the (non-fortuitous) analogy of this definition with:
number \( z \) is said to be rational if and only if there exist two integers \( p \) and \( q \) such that \( z = p/q \).
Proposition 4.12. Relation $R$ is regular on $S$ if and only if it is rational.

Proof. Proof of sufficiency: Let $R$ be written as $R = h \cdot g^\dagger$. Then

$$R \cdot R^\dagger \cdot R = h \cdot g^\dagger \cdot g \cdot h^\dagger \cdot h \cdot g^\dagger$$

$$\leq h \cdot h^\dagger \cdot h \cdot g^\dagger$$

since $g$ is deterministic

$$\leq h \cdot g^\dagger$$

since $h$ is deterministic

$$= R$$

by assumption.

Then, $R$ is regular.

Proof of necessity: Let $R$ be regular, and let $k_R$ and $k'_R$ be the following functions:

$$k_R = \{(s, p) | p = s \cdot R\}, \quad k'_R = \{(s, p) | p = R \cdot s\}.$$

By definition, $k_R$ maps each element of $\text{dom}(R)$ into its image set by $R$, and $k'_R$ maps each element of $\text{rng}(R)$ into its antecedent set by $R$. We define the following relation on $2^S$ (the set of subsets of $S$).

$$f = \{(p, p') | \exists s \in p, \exists s' \in p': (s, s') \in R\}.$$

It is useful to note that $f$ can be written simply as $k_R^\dagger \cdot R \cdot k'_R$. We pose

$$h = k_R \cdot f, \quad g = k'_R.$$

We must prove two lemmas:

(i) $h$ is a function,

(ii) $R = h \cdot g^\dagger$.

Proof of (i):

$$h^\dagger \cdot h = f^\dagger \cdot k_R^\dagger \cdot k_R \cdot f$$

$$\leq f^\dagger \cdot f$$

determinacy of $k_R$

$$\leq I$$

determinacy of $f$.

The proof of the determinacy of $f$ depends on the following property: $k_R^\dagger \cdot R$ is deterministic, due to the definition of $k_R$ and to the regularity of $R^\dagger$.

Proof of (ii):

$$h \cdot g^\dagger$$

$$= k_R \cdot f \cdot k'_R$$

definition of $h$ and $g$

$$= k_R \cdot k_R^\dagger \cdot R \cdot k'_R \cdot k'_R$$

by definition of $f$

$$= (R \cdot R^\dagger) \cdot R \cdot (R^\dagger \cdot R)$$

by the definition of $k_R$, $k_R \cdot k_R^\dagger = R \cdot R^\dagger$

$$= R$$

regularity of $R$. \qed
The proof above has not only established the decomposition of $R$ as $h \ast g^\ast$, it has also, incidentally, given us a decomposition that is canonical in some sense. We discuss this matter further, below.

In [9, 10], relative injectivity of functions was defined as follows: function $f$ is said to be more injective than $f'$ if and only if

$$f' \ast f'^\ast \subseteq f' \ast f'^\ast.$$ 

Using this definition, and letting $k_R$, $k'_R$ and $f$ be defined as in the proof above, we claim the following.

**Proposition 4.13.** Among the decompositions of a regular relation as $h \ast g^\ast$, the following decomposition is optimal, in the sense that its functions are most injective:

$$h = k_R \ast f, \quad g = k'_R.$$ 

**Proof.** Let $R$ be written as $R = H \ast G^\ast$. We must show that $h$ is more injective than $H$, and $g$ is more injective than $G$. Let $(s, s')$ be an element of $h \ast h^\ast$. Then

$$(s, s') \in k_R \ast f \ast f'^\ast \ast k_R^\ast$$

definition of $h$,

$$(s, s') \in k_R \ast k_R^\ast$$

injectivity of $f$, and dom$(f) = \text{rng}(k_R)$,

$$(s, s') \in R \ast R^\ast$$

definition of $k_R$,

$$(s, s') \in H \ast G^\ast \ast G \ast H^\ast$$

decomposition of $R$,

$$(s, s') \in H \ast H^\ast$$

determinacy of $G$.

Hence $h$ is more injective than $H$. Similarly, let $(s, s')$ be an element of $g \ast g^\ast$. Then

$$(s, s') \in k'_R \ast k'_R^\ast$$

definition of $g$,

$$(s, s') \in R^\ast \ast R$$

definition of $k'_R$,

$$(s, s') \in G \ast H^\ast \ast H \ast G^\ast$$

decomposition of $R$,

$$(s, s') \in G \ast G^\ast$$

determinacy of $H$.

Hence $g$ is more injective than $G$. □

A function is all the more injective the more information its image gives about its argument. It is known that if function $f$ is more injective than $f'$ then there exists a function $k$ such that

$$f' = f \ast k.$$ 

We have proven that for any decomposition of $R$ under the form $R = H \ast G^\ast$, $H$ and $G$ are related to the minimal $h$ and $g$ by the following equations: There exists a function $k$ such that

$$H = h \ast k, \quad G = g \ast k.$$
5. Closure of regular relations

In the same way as the property of transitivity yields the notion of transitive closure, and the property of reflexivity yields the notion of reflexive closure, the property of regularity yields the notion of regular closure. In this section, we discuss regular closure. By analogy with other closures, we define it as the smallest regular relation containing a given relation $R$.

The regular closure of relation $R$ is the relation, denoted by $R^\omega$ and defined by:

(i) $R \subseteq R^\omega$,

(ii) $R^\omega$ is regular,

(iii) for all regular relations $T$ that contains $R$, $R^\omega \subseteq T$.

We have a constructive formula for the regular closure.

**Proposition 5.1.** The regular closure of $R$ is given by

$$R^\omega = R \cup R \ast (R^\ast \ast R) \cup R \ast (R^\ast \ast R)^2 \cup \cdots \cup R \ast (R^\ast \ast R)^n \cup \cdots.$$ 

**Proof.** Clause (i) of the definition is trivial. Clause (ii):

$$R^\ast \ast R^\omega = (\bigcup_{i \geq 0} R \ast (R^\ast \ast R)^i) \ast (\bigcup_{i \geq 0} R \ast (R^\ast \ast R)^i)^+$$

$$= \bigcup_{i \geq 0, j \geq 0} (R \ast (R^\ast \ast R)^i) \ast (R \ast (R^\ast \ast R)^i)^+ \ast (R \ast (R^\ast \ast R)^j)^+$$

$$= \bigcup_{i \geq 0, j \geq 0, k \geq 0} (R \ast (R^\ast \ast R)^{i+j+k+1})$$

$$= \bigcup_{j \geq 1} (K \ast (K^\ast \ast R)^j)$$

$$\subseteq R^\omega.$$ 

Clause (iii): Let $T$ be a regular relation containing $R$. The proof that $R^\omega \subseteq T$ can be done by induction on the terms of $R^\omega$. The basis of induction is the hypothesis that $T$ contains $R$. The induction step stems from the regularity of $T$. 

6. Conclusion: future prospects

In this paper, we have focused on a property that has drawn our attention several times as we work on various aspects of a relational approach to structured programming: regularity. This property has proven to be of interest, for two reasons which are seemingly paradoxical: first, its strength, as measured by the ease of manipulation that it affords us; second, its generality, as measured by the weakness of its sufficient conditions.

One of the most intriguing properties that we have found for regular relations is the capability to write them as $h \ast g^\omega$, where $h$ and $g$ are functions. This is intriguingly similar to the capability of rational numbers to be written as $p/q$ where $p$ and $q$
are integers. Notice, in addition that the decomposition given in Proposition 5.1 is minimal with respect to function composition; this again, is subject to a comparison with rational numbers, which have a minimal decomposition (with respect to multiplication) in terms of two relatively prime natural numbers. Perhaps the omnipresence of regular relations is analogous, then, to the "everywhere density" of rational numbers in the set of real numbers. The analogy between functions and integers, regular relations and rationals, and between relations and reals is currently being investigated. We are hoping to find, thereby, some structural properties of the set of relations.

Another subject of investigation that we are currently pursuing deals with the capability to transform a given relation into a regular relation. Of course, the target regular relation must have some coherence property with the original relation. In our study of structured programming, we have found it necessary, quite often, to transform a specification into a more general (in the sense of problem-solving) specification. In relational terms, this means transforming a relation into a more-defined relation. It turns out that the regular closure of a relation \( R \) is not more general than \( R \). We are currently investigating other means to transform a relation into a regular relation, that is at the same time more-defined.

In the introduction, we have pointed out the importance of binary relations in the study of programming. Because regularity is such an important property for binary relations, regular relations are sure to play an important role in programming in the future.

References


