Science of Computer Programming 18 (1992) 67-106
Elsevier

Heuristics for constructing while loops

Fatma Mili

School of Engineering and Computer Science, Oakland University, Rochester, MI 48309-4401, USA

Ali Mili

Département d'Informatique, Faculté des Sciences de Tunis, Université de Tunis II, 1002 Belvédère, Tunisia

Received April 1990
Revised February 1991

Abstract

Mili, F. and A. Mili, Heuristics for constructing while loops, Science of Computer Programming 18 (1992) 67-106.

We discuss the stepwise construction of iterative programs from specifications, represented by relations. We make an effort to isolate, in the construction of an iterative program, those decisions that are dictated by correctness preservation concerns, from decisions that the programmer is free to make at will.

1. Introduction: the problem and its context

Despite several decades of research, the construction of programs from specifications remains an activity where creativity plays an important role. To be sure, the mathematics of program correctness, as we know them today [19, 18, 11, 23, 10], do give some insight into how programs can be constructed from specifications; but there is more to program construction than understanding program correctness. While techniques of program correctness are capable of recognizing that a generated program is correct, they are unable to generate a correct program from a specification. So that despite our understanding of program correctness, much of the burden of decision making in the program construction process still rests on the shoulders of the programmer.

To address this shortcoming, we propose to investigate the intimate mechanisms that define each decision in the stepwise derivation of a program from a specification. For each decision, we wish to identify what aspects can be automated (or at least

made constructive) and what aspects are left to the programmer's discretion. Also, for the latter aspects, we are interested in determining the bounds within which the programmer may make her decisions.

The mathematical tool that we have selected to carry out our study is the algebra of relations. Rather than elaborate on the merits of this tool, we will present two simple arguments in support of our choice:

- We represent program specifications by *homogeneous* relations, i.e. relations from some set S to itself. At least in principle, the hypothesis of homogeneity causes no loss of generality: a relation from S to T can always be considered as a homogeneous relation on $S \cup T$.
- A number of concepts (operations, properties, ...) that we have encountered in our study, and that we will discuss in this paper, can be formulated quite naturally and crisply in the algebra of relations. Yet we do not know how to represent them, let alone discover them, had we used another notation.

In Section 2 we present the background of our study, by giving some elements of mathematics, then presenting our view of program specification, program correctness, and program construction; also, we show at the end of that section why program construction cannot be carried out systematically, and motivate the need for a heuristic approach. Because of its complexity, and its interest, we concentrate on the construction of while loops; this is the subject of Section 3. After introducing some additional mathematical background, we discuss in this section how to derive a while loop from a deterministic specification (i.e. a function), then from a non-deterministic specification.

Just as it is common, in problem-solving, to generalize a problem before solving it, we may have to generalize a specification before applying the iteration rule to it. Section 4 presents a set of generalization heuristics, for this very purpose. Section 5 discusses the completeness of the network of heuristics, and establishes in fact that it is complete, i.e. whenever a specification has a correct program under the form of a while loop, this network can deliver it. In Section 6 we show the application of the heuristics proposed on a non-trivial (although not too complex) example. In particular, the example we present uses non trivial data structures (such as trees, stacks, ...); the main thesis that we submit in this section is that the heuristics that we present can be used at an arbitrary level of detail, provided the appropriate abstractions are defined. In Section 7 we summarize our results and impressions, and discuss the relationship of our work to others.

2. Program construction by relational manipulation

2.1. Elements of discrete mathematics

Some mathematical background is required for the purposes of this paper. For the sake of readability, we introduce this background little by little, as it is needed;

hence in this subsection we content ourselves with presenting the notions that are needed for Section 2.

A *relation* on set S is a subset of $S \times S$. Constant relations on set S include: the *universal* relation, $L = S \times S$, the *identity* relation $I = \{(s, s) \mid s \in S\}$ and the *empty* relation $\Phi = \{\}$. If A is a subset of S, we let $I(A)$ be the relation defined by:

$$I(A) = \{(s, s) \mid s \in A\}.$$

The *domain* of a relation R is denoted by $dom(R)$, and the range of relation R is denoted by $rng(R)$. A relation whose domain is S is said to be *total.* A relation whose range is S is said to be *surjective.* The *image set* of element s by relation R is:

$$s.R = \{s' \mid (s, s') \in R\}.$$

The *product* of relations R and Q is denoted by $R \circ Q$, or for the sake of compactness, RQ. The *transitive closure* of relation R is denoted by R^+, and the *reflexive transitive closure* by R^*. If T is the transitive closure of R, we say that R is a *transitive root* of T. Given a relation R and a subset A of S, we define the *prerestriction* of R to A to be the relation denoted by ${}_{A|}R$ and defined by $I(A) \circ R$. The *postrestriction* is defined similarly.

A relation R is said to be *more-defined* than relation Q if and only if

$$dom(Q) \subseteq dom(R),$$

$$\forall s \in dom(Q),\ s.R \subseteq s.Q.$$

Intuitively speaking, a relation R is more-defined than a relation Q if and only if it carries more input output information: R knows about more inputs than Q, since it has a larger domain; for inputs about which both know, R is more accurate in its assignment of outputs to inputs.

A relation R is said to be *range identical* if and only if:

$$\forall s \in rng(R),\ s.R = \{s\}.$$

In other words, a relation R is range identical if and only if for all s in $rng(R)$, the pair (s, s) is in R, and no other pair (s, s') for $s \neq s'$ is in R. This can also be expressed by the equality

$$I(rng(R)) \circ R = I(rng(R)).$$

For example, the relation

$$\{(6, 0), (5, 1), (4, 0), (3, 1), (2, 0), (1, 1), (0, 0)\}$$

is range identical, whereas the relations

$$\{(6, 0), (5, 1), (4, 0), (3, 1), (2, 0), (1, 1), (1, 0), (0, 0)\},$$

$$\{(6, 0), (5, 1), (4, 0), (3, 1), (2, 0), (1, 1)\}$$

are not, because $I(rng(R)) \circ R$ is larger than $I(rng(R))$ in the first case, smaller in the second case.

2.2. Program specification, program correctness

We consider the following Pascal-like variable declarations:

x: **integer**;

y: **real**;

given such a declaration, we define a *state* to be a pair $\langle x, y \rangle$ such that x is of type **integer** and y is of type **real**; for a given state, say s, we let $x(s)$ and $y(s)$ denote the x-component and y-component of state s. The set of all the states, for all possible values of $x(s)$ and $y(s)$ is called a *space*.

We have defined a specification when we have given: a *space* S; and a *relation* R on S.

A program P on space S defines a function on S, made up of the set of ⟨initial state, final state⟩ pairs that it defines [18, 23]. This function we denote by $[P]$. A program P on space S is said to be *correct* with respect to specification R on S if and only if $[P]$ is more-defined than R. This definition is identical to traditional definitions of total correctness [19].

2.3. Program construction by relational decomposition

A specification is a relation; hence the construction of a program from a specification proceeds by a stepwise transformation of complex relations into more tractable relations. We articulate this process as follows:

construct(R): If R is simple then
  find an assignment statement correct with respect to R
else
  transform R into $R_1, R_2, \ldots, R_n$
  apply procedure *construct* to $R_1, R_2, \ldots, R_n$.

2.4. Construction rules

The question that the procedure given above raises immediately is: How do we transform a given specification R into specifications $R_1, R_2, \ldots, R_n$, and indeed what relationship must there be between the original specification and the derived specifications. We have identified two kinds of transformations.

- *Decomposition* transformations, which map a given specification into one or more simpler (in some sense) specifications; these transformations rewrite specification R as a relational expression (using relational operations) involving $R_1, R_2, \ldots, R_n$. The link between the original expression and the derived relational expression is one of *equality*.
- The *generalization*, which maps a given specification into a more-defined (intuitively: more general) specification.

We discuss below the types of rules that govern these transformations.

Decomposition rules

We have identified three such rules, which correspond to three relational operators, and (not incidentally) to three Pascal constructs.

Sequence Rule. Given a relation R, find R_1 and R_2 such that

$$R = R_1 \circ R_2 \ \wedge \ rng(R_1) \subseteq dom(R_2).$$

Proposition. *If p_1 is correct with respect to R_1 and p_2 is correct with respect to R_2 then*

begin p_1 ; p_2 **end**

is correct with respect to R.

Alternation Rule. Given a relation R, find relations R_1 and R_2 such that

$$R = R1 \cup R2 \ \wedge \ dom(R1) \cap dom(R2) = \emptyset.$$

Proposition. *Let t be defined by $t(s) = s \in dom(R_1)$. If p_1 is correct with respect to R_1 and p_2 is correct with respect to R_2 then*

if t **then** p_1 **else** p_2

is correct with respect to R.

Iteration Rule (original version due to Mills et al. [23]). Given a relation R which is total and range identical, find a relation B such that B^+ is a well-founded ordering, and such that

$$R = B^* \circ I(rng(R)) \ \wedge \ dom(B) = dom(R) - rng(R).$$

Proposition. *Let t be defined by $t(s) = s \in dom(B)$. If b is correct with respect to B then*

while t **do** b

is correct with respect to R. If R is not total or is not range identical, then there exists no relation B which is a transitive root of a well-founded ordering and such that

$$R = B^* \circ I(rng(R)) \ \wedge \ dom(B) = dom(R) - rng(R).$$

The generalization rule

This rule is the programmer's version of the well-known problem solving pattern of *generalization* [17].

Generalization Rule. Given a relation R, find a relation R' that is more-defined than R.

Proposition. *If p is correct with respect to R' then p is correct with respect to R.*

2.5. Program construction heuristics

Program construction is a creative activity; hence giving the rules that govern its steps is hardly of any value to the programmer in taking correct steps; rather it is only useful in checking steps after they have been taken. Hence, we need to propose *heuristics*, in the form of constructive procedures, that help the programmer make her design decisions in a systematic fashion.

Naturally, we have focused our attention on the iteration rule, and have derived heuristics that are instances of this rule. Now we have found it common that a given specification admits a solution under the form of a while loop, yet cannot be processed by the given heuristics; recognizing that for such specifications the generalization rule must be applied before the iteration heuristics, we have derived several heuristics which are instances of the generalization rule.

For the sake of readability, we define some notational conventions that we use uniformly throughout this paper: The iteration heuristics will be denoted by capital T, with indices; the generalization heuristics will be denoted by capital G, with indices. Specifications of loop bodies will be denoted by capital B, possibly indexed; specifications of while statements will be denoted by capital W, possibly indexed.

3. Heuristics for iteration

We consider the iteration rule again:

> Given a relation W which is total and range identical, find a relation B such that B^+ is a well-founded ordering, and that
>
> $$W = B^* \circ I(rng(W)) \quad \wedge \quad dom(B) = dom(W) - rng(W).$$

The question that we wish to address in this section is: how do we derive B from W? We will discuss this matter in the case when specification W is deterministic, then the case when it is not. First, we give some elements of mathematics.

3.1. Nuclei and kernels

The *nucleus* of relation R is the relation denoted by $\nu(R)$ and defined by: $\nu(R) = R\hat{R}$. The nucleus of relation R contains all the pairs (s, s') such that s and s' share a common image by R.

The *kernel* of relation R is the relation denoted by $\kappa(R)$ and defined by $\kappa(R) = \{(s, s') | \emptyset \neq s'.R \subseteq s.R\}$. A possible interpretation we could give of this operation is that $\kappa(R)$ is the least defined solution (in X), when it exists, of the system of equations:

$$XR = R,$$

$$X \subseteq L\hat{R}.$$

Hence $\kappa(R)$ is the *weakest prespecification* of R by itself, in the sense of Hoare and He [13].

A relation R is said to be *regular* if and only if $R = R\hat{R}R$. This notion is known to Riguet [26] under the name *difonctionelle* and to Berghammer [7] under the derived name *difunktional.* Its pertinence to programming is highlighted in [15], where a number of sufficient conditions for regularity are given. For the purposes of this paper, it suffices to note two details: all specifications we will be using in this paper are regular; when a specification is regular, its kernel is identical to its nucleus.

3.2. A heuristic for deterministic specifications

Given a relation W which is total, deterministic and range identical, we seek a systematic way to decompose it by the iteration rule. The heuristic given below proposes a solution, i.e., heuristic $T1$ extracts a loop body specification from the loop specification.

Heuristic $T1$. Given a relation W which is total, deterministic and range identical, choose a well-founded ordering GT such that

$$I(S - rng(W)) \circ W \subseteq GT,$$

then pose

$$B = (I(S - rng(W)) \circ \nu(W)) \cap GT.$$

Three comments are in order about this heuristic:

- First, it is rather restrictive, since it only applies to specifications that are total, deterministic, and range identical.
- Second, perhaps consequently, it is very imperative in determining the proposed solution; except for the choice of GT, the solution proposed for B is explicitly constructed in terms of W. As the example below illustrates, the choice of the well-founded ordering GT can hardly be considered arbitrary; rather it is itself very restricted by the condition

  $$I(S - rng(W)) \circ W \subseteq GT,$$

  hence adding to the imperative, constructive nature of this heuristic.

- Third, there always exists a well-founded relation GT that satisfies the requirements of this heuristic: $GT_0 = I(S - rng(W)) \circ W$. It suffices to observe that the product of this relation by itself is the empty relation, hence: this relation is vacuously transitive, and has no infinite decreasing sequences (its longest chain is of length one). We may, in practice, want to choose larger (with respect to inclusion) solutions for GT, so as to get smaller steps for the loop body.

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

a: **array** $[1..n]$ **of real**;

i: $1..n+1$;

x: **real**

and let W be the specification defined as:

$$W = \left\{ (s, s') \;\middle|\; a(s') = a(s) \wedge i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}.$$

W is total (no restriction on s), deterministic (all of $a(s')$, $i(s')$ and $x(s')$ are specified), and range identical. We briefly check the latter condition.

$$rng(W) = \{s \mid i(s) = n+1\}.$$

$$I(rng(W)) \circ W$$

$$= \left\{ (s, s') \;\middle|\; i(s) = n+1 \wedge a(s') = a(s) \wedge i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}$$

$$= \{(s, s') \mid i(s) = n+1 \wedge a(s') = a(s) \wedge i(s') = i(s) \wedge x(s') = x(s)\}$$

$$= I(rng(W)).$$

Application of $T1$ yields:

$$I(S - rng(W)) \circ W$$

$$= \left\{ (s, s') \;\middle|\; i(s) \neq n+1 \wedge a(s') = a(s) \wedge i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}$$

$$= \left\{ (s, s') \;\middle|\; i(s) < n+1 \wedge a(s') = a(s) \wedge i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}$$

$$= \left\{ (s, s') \;\middle|\; i(s) < i(s') \wedge a(s') = a(s) \wedge i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}.$$

We must select a well-founded ordering GT that is a superset of the above relation. To take a superset, it suffices to select a subset of the conjuncts defining this relation. Among the four conjuncts, the only one that defines a well-founded ordering is: $i(s) < i(s')$. Hence, we must include it in GT. We choose

$$GT = \{(s, s') \mid i(s) < i(s')\}.$$

Whence

$$B = \left\{ (s, s') \;\middle|\; i(s) < n+1 \wedge a(s) = a(s') \wedge x(s) + \sum_{k=i(s)}^{n} a(s)[k] = x(s') + \sum_{k=i(s')}^{n} a(s')[k] \wedge i(s) < i(s') \right\}.$$

The work of heuristic $T1$ is finished here, when B is delivered. For the sake of illustration, we analyse the loop body specification that it proposes. The loop body is only applicable if $i(s) < n+1$; when such is the case, the loop body must

- preserve array a,
- increase index i,
- preserve the expression $x(s) + \sum_{k=i(s)}^{n} a(s)[k]$.

If it is decided that index i increases by 1 at each iteration, then:

- we do not modify a (to preserve a),
- we perform $i := i+1$ (to increase i),
- we perform $x := x + a[i]$ (to preserve $x(s) + \sum_{k=i(s)}^{n} a(s)[k]$).

This yields the following while statement

while $i < n+1$ **do**
  begin
  $i := i+1$ {increase i by 1};
  $x := x + a[i]$ $\left\{ \text{preserves } x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}$
  end.

This program is correct with respect to W. □

Remark. Notice that in this heuristic, we can replace the clause *deterministic* by the clause *regular*, without affecting the validity of the results. Indeed we have found that all the results we have used in [21] to get this heuristic hold for regular relations. Unfortunately, we have found that any regular relation that is range identical is deterministic. Hence replacing the clause *deterministic* by the clause *regular* would not make our heuristic any more general. □

3.3. A heuristic for non-deterministic specifications

Heuristic $T1$ is only applicable to deterministic specifications; because one is often faced with non-deterministic specifications (where e.g. not all final values of variables are specified), we must find means to handle non-deterministic specifications. We present heuristic $T2$, for this purpose. First, we give some results that help establish the validity of this heuristic.

A relation W is said to be *range inclusive* if and only if for all s in $rng(W)$, $(s, s) \in W$. We admit without proof that a relation W is range inclusive if and only if $W \circ (W \cap I) = W$.

Lemma (Heuristic $T2$). *Let W be a total relation which is range inclusive. Then*

$$\kappa(W) \circ I(rng(W)) \subseteq W.$$

Proof. The definition of the range inclusive property provides that if W is range inclusive then $I(rng(W)) \subseteq W$. We compute $\kappa(W) \circ I(rng(W))$, making use of this identity.

$$\kappa(W) \circ I(rng(W)) = \{(s, s') \mid \emptyset \neq s'.W \subseteq s.W \wedge (s', s') \in I(rng(W))\}$$

by definition of $\kappa(\ )$, and of $I(rng(W))$

$$= \{(s, s') \mid \emptyset \neq s'.W \subseteq s.W \wedge (s', s') \in W\}$$

by hypothesis, and definition of range inclusive

$$= \{(s, s') \mid \emptyset \neq s'.W \subseteq s.W \wedge s' \in s'.W\}$$

by definition of image sets

$$\subseteq \{(s, s') \mid s' \in s.W\}$$

logical consequence of: $s'.W \subseteq s.W \wedge s' \in s'.W$.

$$= W \quad \text{by definition.}$$ □

Using this lemma, we establish the following theorem.

Theorem (Heuristic $T2$). *Let W be a relation on S, which is total and range inclusive, and such that $\kappa(W) \circ I(rng(W))$ is total and let GT be a well-founded ordering relation such that*

$$I(S - rng(W)) \circ W \subseteq GT.$$

Then

$$V = (I(S - rng(W)) \circ \kappa(W) \cap GT) \circ I(rng(W)) \cup I(rng(W))$$

is more-defined than W.

Proof. In order to show that V is more-defined than W, and because W is total, we must show:

(i) $dom(V) = S$

(ii) $V \subseteq W$.

Proof of (i).

$$V = (I(S - rng(W)) \circ \kappa(W) \cap GT) \circ I(rng(W)) \cup I(rng(W))$$

by the formula of V

$$= (I(S - rng(W)) \circ \kappa(W) \circ I(rng(W))) \cap (GT \circ I(rng(W)))$$

$$\cup I(rng(W))$$

by the identity $(R \cap R') \circ I(A) = R \circ I(A) \cap R' \circ I(A)$.

$$\supseteq (I(S - rng(W)) \circ \kappa(W) \circ I(rng(W)))$$

$$\cap (I(S - rng(W)) \circ W \circ I(rng(W))) \cup I(rng(W))$$

by construction of GT

$$= I(S - rng(W)) \circ \kappa(W) \circ I(rng(W)) \cup I(rng(W)).$$

To justify this last step, we observe that

$$\kappa(W) \circ I(rng(W)) \subseteq W = W \circ I(rng(W)),$$

therefore, by prerestriction,

$$I(S - rng(W)) \circ \kappa(W) \circ I(rng(W)) \subseteq I(S - rng(W)) \circ W \circ I(rng(W));$$

hence the intersection of these two terms equals the former. We focus now on the domain of V.

$$dom(V) \supseteq dom(I(S - rng(W)) \circ \kappa(W) \circ I(rng(W))) \cup rng(W)$$

domain of a union on the equation above,

and because $dom(I(A)) = A$

$$= (S - rng(W)) \cup rng(W)$$

$dom(I(A) \circ R) = A \cap dom(R)$, and due to the hypothesis,

$dom(\kappa(W) \circ I(rng(W))) = S$

$= S$ by set-theoretic identity.

Proof of (ii).

$$V = (I(S - rng(W)) \circ \kappa(W) \cap GT) \circ I(rng(W)) \cup I(rng(W))$$

by definition

$$\subseteq (I(S - rng(W)) \circ \kappa(W)) \circ I(rng(W)) \cup I(rng(W))$$

by the identity $R \cap R' \subseteq R$, and the monotony of $\circ$ and $\cup$

$$= I(S - rng(W)) \circ \kappa(W) \circ I(rng(W) \cup I(rng(W))$$

by associativity

$$\subseteq I(S - rng(W)) \circ W \cup I(rng(W))$$

by the lemma above

$$\subseteq W$$

since both terms are subsets of W: the first by its very construction; the second by hypothesis (W is range inclusive). □

The interest of this theorem is the following: because V is more-defined than W, we can substitute the resolution of W by the resolution of V; on the other hand, because V can be written as

$$(I(S - rng(W)) \circ \kappa(W) \cap GT) \circ I(rng(W)) \cup I(rng(W))$$

then

$$B = I(S - rng(W)) \circ \kappa(W) \cap GT$$

is a possible solution of the equation

$$V = B^+ \circ I(rng(W)) \cup I(rng(W)).$$

Indeed, we have, by substitution,

$$V = B \circ I(rng(W)) \cup I(rng(W)).$$

Also, because B is transitive (intersection of two transitive relations), it equals its own transitive closure. Hence B can be obtained from V by the iteration rule, while V can be obtained from W by the generalization rule.

Remark. Strictly speaking, the derivation of B from V cannot be considered an instance of the iteration rule, unless we prove that V verifies the conditions of this rule, namely:

(i) $dom(V) = S$,

(ii) $I(rng(V)) \circ V = I(rng(V))$.

The first property is established in the proof of the theorem above; the second condition can be checked by proving $rng(V) = rng(W)$, then, $I(rng(W)) \circ V = I(rng(W))$. □

We now give heuristic $T2$, which handles non-deterministic specifications.

Heuristic $T2$. Given a relation W on S, which is total, range inclusive, and such that $\kappa(W) \circ I(rng(W))$ is total, choose a well-founded ordering GT such that

$$I(S - rng(W)) \circ W \subseteq GT,$$

then pose

$$B = I(S - rng(W)) \circ \kappa(W) \cap GT.$$

This heuristic, like $T1$, is quite imperative; it gives the solution B as an explicit expression of specification W and the well-founded ordering GT. Also, specification GT is itself subject to such a restrictive condition that in practice there is little latitude in making this choice. Not surprisingly, heuristic $T2$ is a generalization of heuristic $T1$: if we apply $T2$ to a deterministic, total relation W which is range identical, we find the same result as if we applied $T1$. We give here a brief argument to this effect: Because W is range identical, it is range inclusive. Because W is deterministic, $\kappa(W) = \nu(W)$. Because W is range identical, $W = \nu(W) \circ I(rng(W))$. Because W is total, so is $\nu(W) \circ I(rng(W))$. Hence W satisfies all the conditions of $T2$. Because $\kappa(W) = \nu(W)$, the proposed solution B is identical in both heuristics. Before we illustrate this heuristic with an example, we mention an important detail about it.

Remark. The solution B proposed in this heuristic is not a decomposition of W by the iteration rule; rather it is a decomposition of V, where V is more-defined than W. Hence strictly speaking, heuristic $T2$ is not an instance of the iteration rule, but a combined instance of the generalization followed by the iteration rule. Hence it is by abuse of convention that we consider $T2$ to be an instance of the iteration rule. □

Example. We let S be the space defined by the following variable declarations,

a: **array** $[1..n]$ **of real**;

i: $1..n+1$;

x: **real**;

f: **boolean**,

and we let W be the following specification of a search routine,

$$W = \{(s, s') \mid i(s') = n+1 \wedge f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n]))\}.$$

Clearly, W is total. Note that it is not deterministic ($x(s')$ and $a(s')$ are not specified) hence we have no temptation to apply $T1$. We check whether it is range inclusive.

To do so, we compute

$$
\begin{aligned}
& W \circ (W \cap I) \\
& \quad = \{(s, s') \mid i(s') = n+1 \wedge f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n]))\} \\
& \qquad \circ \{(s, s') \mid i(s) = n+1 \wedge f(s) = (f(s) \vee (x(s) \in a(s)[i(s)..n])) \wedge s' = s\} \\
& \qquad\qquad \text{by definition} \\
& \quad = \{(s, s') \mid i(s') = n+1 \wedge f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n]))\} \\
& \qquad \circ \{(s, s') \mid i(s) = n+1 \wedge s' = s\} \\
& \qquad\qquad \text{if } i(s) = n+1 \text{ then } a(s)[i(s)..n] \text{ is empty,} \\
& \qquad\qquad \text{and } (x(s) \in a(s)[i(s)..n]) = \textbf{false}, \\
& \qquad\qquad \text{hence } (f(s) = (f(s) \vee (x(s) \in a(s)[i(s)..n]))) = \textbf{true} \\
& \quad = \{(s, s') \mid i(s') = n+1 \wedge f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n])) \\
& \qquad\qquad \wedge i(s') = n+1\} \\
& \qquad\qquad \text{by definition of product (in this case, post-restriction)} \\
& \quad = \{(s, s') \mid i(s') = n+1 \wedge f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n]))\} \\
& \qquad\qquad \text{by logical identity} \\
& \quad = W \qquad \text{by definition of } W.
\end{aligned}
$$

Hence W is indeed range inclusive. We must check whether $\kappa(W) \circ I(rng(W))$ is total. Relation W is regular, because it can be written as the product of relation

$$
\begin{aligned}
& \{(s, s') \mid a(s') = a(s) \wedge x(s') = x(s) \wedge i(s') = n+1 \\
& \qquad \wedge f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n]))\},
\end{aligned}
$$

which is a function, by relation

$$
\{(s, s') \mid i(s') = i(s) \wedge f(s') = f(s)\},
$$

which is an equivalence relation, hence a regular relation.[1] Because W is regular, its kernel is identical to its nucleus. Hence

$$
\begin{aligned}
& \kappa(W) \circ I(rng(W)) \\
& \quad = \nu(W) \circ I(rng(W)) \\
& \quad = \{(s, s') \mid (f(s) \vee (x(s) \in a(s)[i(s)..n])) \\
& \qquad\qquad = (f(s') \vee (x(s') \in a(s')[i(s')..n])) \wedge i(s') = n+1\} \\
& \quad = \{(s, s') \mid f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n])) \wedge i(s') = n+1\}.
\end{aligned}
$$

[1] We have shown in [15] that the product of a function by a regular relation yields a regular relation, and that an equivalence relation is regular.

This is clearly a total relation: for any given s, one can always take

$$f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n])),$$

$$a(s') = a(s), \qquad x(s') = x(s), \quad \text{and} \quad i(s') = n+1.$$

We now compute $I(S - rng(W)) \circ W$, then take a superset of it as GT:

$$I(S - rng(W)) \circ W$$

$$= \{(s, s') \mid i(s) \neq n+1 \wedge i(s') = n+1 \wedge f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n]))\}$$

by definition of restriction

$$= \{(s, s') \mid i(s) < n+1 \wedge i(s') = n+1 \wedge f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n]))\}$$

because i is declared of type: $1..n+1$

$$= \{(s, s') \mid i(s) < i(s') \wedge i(s') = n+1 \wedge f(s') = (f(s) \vee (x(s) \in a(s)[i(s)..n]))\}$$

because $i(s') = n+1$.

To take a well-founded ordering that is a superset of this relation, we select a subset of the conjuncts that defines a well-founded ordering. The first conjunct is adequate.

$$GT = \{(s, s') \mid i(s) < i(s')\}.$$

The loop body specification that stems from this choice of GT is:

$$B = I(S - rng(W)) \circ \kappa(W) \cap GT$$

by heuristic $T2$

$$= I(S - rng(W)) \circ \nu(W) \cap GT$$

by regularity of W

$$= \{(s, s') \mid i(s) \neq n+1 \wedge (f(s) \vee (x(s) \in a(s)[i(s)..n])) = (f(s') \vee (x(s') \in a(s')[i(s')..n])) \wedge i(s) < i(s')\}.$$

The job of heuristic $T2$ is finished here, with the delivery of specification B. For the sake of further illustration, however, we show an example of subsequent development of the loop body specification B. This specification prescribes that the loop body is invoked only if $i(s) \neq n+1$ (i.e. $i(s) < n+1$, due to the type of i). When it is invoked, the loop body must increase index i, while preserving the expression

$$f(s) \vee (x(s) \in a(s)[i(s)..n]).$$

Hence, for example, if i is increased by 1, the variable f must be updated by the statement:

$$f := f \quad \text{or} \quad (x = a[i]).$$

This yields the following while statement:

while $i \neq n+1$ **do**
  begin
  $i := i+1$;
  $f := f$ or $x = a[i]$
  end,

which is correct with respect to W. □

4. Heuristics for generalization

It is common to find a specification that is known to have a solution under the form

while t **do** b,

and yet does not meet the applicability conditions of heuristics $T1$ and $T2$ (which are rather stringent). An example of such a specification is the relation W defined on space $S = \{0, 1, 2, 3, 4, 5, 6\}$ by,

$$W = \{(6, 0), (5, 1), (4, 0), (3, 1), (2, 0), (1, 1)\}.$$

While we know well that this specification can be satisfied by a while loop, namely

while $s > 1$ **do** $s := s - 2$,

we find that we cannot apply heuristic $T1$, nor heuristic $T2$, because W is not even total (not to mention that it is neither range inclusive, nor, consequently, range identical).

In such cases, (instances of) the generalization rule must be applied before the iteration heuristics; this problem-solving step is exactly identical to generalizing a problem definition before carrying out an inductive argument (see [17], for a general problem-solving presentation; and [3, 11, 25] for a programming oriented presentation).

The purpose of this section is to present heuristics that, given a specification W which is known to have a while statement as a solution but does not meet the conditions of applicability of $T1$ nor those of $T2$, will map W into a specification W' that meets the conditions of $T1$ or $T2$ and is more defined (i.e. more general) than W. As we mentioned above, these heuristics are instances of the generalization rule. Before we present them, we give some mathematical background.

4.1. Iterative forms

In investigating properties of relations which have a while loop as a solution, we have identified four classes of these relations, ordered by inclusion. In this section we define these four classes, and give some intuitive feel for them; in order to illustrate these classes, we give two running examples. As a reminder, we restate

that a relation W is said to be range identical if and only if $I(rng(W)) \circ W = I(rng(W))$. Range identical relations behave like the identity relation on their range; from a programming standpoint, they can be characterized by the fact that they admit a decomposition by the iteration rule. A relation W is said to be *4-iterative* if and only if it is total and range identical.

Example (4-iterative property). Let S be the space defined by $S = \{0, 1, 2, 3, 4, 5, 6\}$ and let W be the relation defined by:

$$W = \{(6, 0), (5, 1), (4, 0), (3, 1), (2, 0), (1, 1), (0, 0)\}.$$

This relation is 4-iterative for it is total and range identical. We leave it to the reader to check that it can indeed be written as

$$W = B^* \circ I(rng(W)),$$

for $B = \{(6, 4), (5, 3), (4, 2), (3, 1), (2, 0)\}$. Note also that B satisfies the other requirments of the iteration rule.

Let S be the space defined by the following declarations,

a, b, c: **natural**,

and such that $a \neq 0$. We let W be the following relation on S:

$$W = \{(s, s') \mid a(s') = a(s) \wedge b(s') = 0 \wedge c(s') = c(s) + a(s)b(s)\}.$$

This relation is clearly total; on the other hand, its range is $\{s \mid b(s) = 0\}$ and its prerestriction to its range is

$$\begin{aligned} I(rng(W)) \circ W &= \{(s, s') \mid b(s) = 0 \wedge a(s') = a(s) \wedge b(s') = b(s) \wedge c(s') = c(s)\} \\ &= I(rng(W)). \end{aligned}$$

Hence it is 4-iterative. □

We restate that a relation W is said to be range inclusive if and only if $I(rng(W)) \subseteq W$, and we admit without proof that this condition is equivalent to $W \circ (W \cap I) = W$ (the latter condition is usually easier to check). Clearly, if a relation is range identical then it is range inclusive. A relation is said to be *3-iterative* if and only if it is total and range inclusive.

Example (3-iterative property). Let S be the space defined by $S = \{0, 1, 2, 3, 4, 5, 6\}$ and let W be the relation defined by

$$W = \{(6, 0), (5, 1), (4, 0), (3, 1), (2, 0), (1, 1), (1, 0), (0, 1), (0, 0)\}.$$

This relation is total; on the other hand, it is range inclusive for it contains $\{(1, 1), (0, 0)\}$, which is $I(rng(W))$. Hence it is 3-iterative.

Let S be the space defined by the following declarations,

a, b, c: **natural**,

and such that $a \neq 0$. We let W be the following relation on S:

$$W = \{(s, s') \mid b(s') = 0 \wedge c(s') = c(s) + a(s)b(s)\}.$$

Relation W is clearly total; we check that it is range inclusive:

$$\begin{aligned} & W \circ (W \cap I) \\ & \quad = \{(s, s') \mid b(s') = 0 \wedge c(s') = c(s) + a(s)b(s)\} \\ & \qquad \circ \{(s, s') \mid s' = s \wedge b(s') = 0 \wedge c(s') = c(s) + a(s)b(s)\} \\ & \quad = \{(s, s') \mid b(s') = 0 \wedge c(s') = c(s) + a(s)b(s)\} \circ \{(s, s') \mid b(s) = 0 \wedge s' = s\} \\ & \quad = W \circ I(rng(W)) = W. \end{aligned}$$

Hence W is 3-iterative. □

A relation W is said to be *range generative* if and only if W and $W \circ (W \cap I)$ have the same domain. Intuitively speaking, a relation is range generative if and only if we can reduce its range, by generalization, to make it range inclusive. Clearly, if a relation is range inclusive then it is range generative. A relation is said to be *2-iterative* if and only if it is total and range generative.

Example (2-iterative property). Let S be the space defined by $S = \{0, 1, 2, 3, 4, 5, 6\}$ and let W be the following relation on S:

$$W = \{(6, 2), (6, 0), (5, 3), (5, 1), (4, 4), (4, 0), (3, 3), (3, 1), (2, 0), (1, 1), (0, 0)\}.$$

This relation is total; we check that it is also range generative.

$$\begin{aligned} W \circ (W \cap I) &= W \circ \{(4, 4), (3, 3), (1, 1), (0, 0)\} \\ &= \{(6, 0), (5, 3), (5, 1), (4, 4), (4, 0), (3, 3), (3, 1), (2, 0), (1, 1), (0, 0)\}. \end{aligned}$$

This relation has the same domain as W; hence W is range generative. Because it is also total, it is 2-iterative.

Let S be the space defined by the following declarations,

a, b, c: **natural**,

and such that $a \neq 0$ and let W be the following relation on S:

$$W = \{(s, s') \mid c(s') = c(s) + a(s)b(s)\}.$$

This relation is total. We check whether it is range generative. To do so, we compute,

$$W \circ (W \cap I) = W \circ \{(s, s') \mid c(s) = c(s) + a(s)b(s) \wedge s' = s\}$$

by definition

$$= W \circ \{(s, s') \mid b(s) = 0 \wedge s' = s\}$$

because $a \neq 0$

$$= \{(s, s') \mid c(s') = c(s) + a(s)b(s) \wedge b(s') = 0\}$$

by postrestriction.

Because this relation has the same domain as W (both are total), W is range generative.[2] Because W is also total, it is 2-iterative. □

A relation W is said to be *iterative* (or *1-iterative*) if and only if it has the same domain as

$$W \circ ((W \cap I) \cup I(S - dom(W))).$$

Clearly, if a relation is range generative then it is iterative. The postrestriction of W, by multiplication on the right by $W \cap I$, can reduce the domain of W if and only if there exists s in $dom(W)$ such that for all s' in the image set of s by W, the pair (s', s') is not in W; the iterative property provides that for those s there must exist at least one s' in the image set such that $s' \notin dom(W)$. If we may give, before proving it, the interpretation of this property from a programming standpoint, a relation W is iterative if and only if it admits a while loop as a correct program. Note that by contrast with all three properties given above, the property of 1-iterative does not require that W be total.

Example (1-iterative property). Let S be the space defined by $S = \{0, 1, 2, 3, 4, 5, 6\}$, and let W be the following relation on S:

$$W = \{(6, 4), (6, 2), (5, 3), (4, 2), (1, 1), (0, 0)\}.$$

Note that W is not total, nor does it have to be total in order to be 1-iterative. We compute

$$W \circ ((W \cap I) \cup I(S - dom(W))) = W \circ (\{(1, 1), (0, 0)\} \cup \{(2, 2), (3, 3)\})$$

$$= W \circ \{(3, 3), (2, 2), (1, 1), (0, 0)\}$$

$$= \{(6, 2), (5, 3), (4, 2), (1, 1), (0, 0)\}.$$

This relation has the same domain as W, although it is different from W. Hence W is 1-iterative. The reader may check that W is not range generative, as the domain of $W \circ (W \cap I)$ is $\{1, 0\}$, much smaller than the domain of W.

[2] Note on this example how the range of relation $W' = W \circ (W \cap I)$ is generated from relation W; this justifies the name given to this property.

Let S be the space defined by the following variable declarations:

a, b, c: **natural**,

and such that $a \neq 0$. We let W be the following relation on S:

$$W = \{(s, s') \mid c(s) > 10 \wedge c(s') = c(s) + a(s)b(s)\}.$$

Note that this relation is not 2-iterative because it is not total. We check whether it is 1-iterative.

$$\begin{aligned}
&W \circ (W \cap I) \cup W \circ I(S - dom(W)) \\
&\quad = W \circ \{(s, s') \mid c(s) > 10 \wedge b(s) = 0 \wedge s' = s\} \cup W \circ I(S - dom(W)) \\
&\quad = \{(s, s') \mid c(s) > 10 \wedge c(s') = c(s) + a(s)b(s) \wedge c(s') > 10 \wedge b(s') = 0\} \\
&\qquad\qquad \cup W \circ I(S - dom(W)) \\
&\quad = \{(s, s') \mid c(s) > 10 \wedge c(s) + a(s)b(s) > 10 \wedge b(s') = 0 \\
&\qquad\qquad \wedge c(s') = c(s) + a(s)b(s)\} \cup W \circ I(S - dom(W)) \\
&\quad = \{(s, s') \mid c(s) > 10 \wedge b(s') = 0 \wedge c(s') = c(s) + a(s)b(s)\} \\
&\qquad\qquad \cup W \circ I(S - dom(W)).
\end{aligned}$$

This relation is the union of two relations, whose first term has a domain equal to $dom(W)$ (and equal to $\{s \mid c(s) > 10\}$) and whose second term has a domain which is by construction smaller than the domain of W. Hence the domain of this relation is exactly equal to the domain of W. Therefore relation W is iterative. □

Given the definitions of this section, we can now express the purpose of the generalization heuristics in a more articulate fashion: the purpose of these heuristics is to raise specifications from the first iterative property to the third or fourth.

4.2. Heuristic G1: making a specification deterministic

First of all, let us review the applicability conditions of heuristics $T1$ and $T2$:

- for $T1$:
  - W is total,
  - W is deterministic,
  - W is range identical.
- for $T2$:
  - W is total,
  - W is range inclusive,
  - $\kappa(W) \circ I(rng(W))$ is total.

We look at the conditions of $T1$. We propose below a heuristic which provides the *determinacy* property, in case the specification at hand is not deterministic.

Heuristic *G*1. Given a total relation W which is range inclusive, pose $V = I(S - rng(W)) \circ W$ and find a sub-relation of V, say V', that has the same domain as V, and is deterministic. Then take $W' = V' \cup I(rng(W))$.

We give without proof the two key properties of this heuristic, namely:

- W' is more-defined than W,
- W' is total, deterministic, and range identical.

The heuristic owes its validity to the first premise; and owes its usefulness to the second premise. We will mention an additional interesting (though not essential) property of this heuristic: Whenever we submit to $G1$ a specification W which is already total, deterministic, and range identical, heuristic $G1$ will preserve it, i.e. return $W' = W$. This is a pleasant sign of minimality.

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

a: **array** [1..n] **of real**;

i: 1..$n+1$;

x: **real**,

where all the cells of array a are positive, and let W be the specification defined as:

$$W = \left\{ (s, s') \;\middle|\; i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}.$$

This specification is clearly total. Because it is not deterministic, we are not tempted to apply $T1$ to it. But we will attempt to apply $G1$. In order to check whether it is range inclusive, we compute

$$\begin{aligned}
&W \circ (W \cap I) \\
&= \left\{ (s, s') \;\middle|\; i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\} \\
&\quad \circ \left\{ (s, s') \;\middle|\; i(s) = n+1 \wedge x(s) = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \wedge s' = s \right\} \\
&= \left\{ (s, s') \;\middle|\; i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \wedge i(s') = n+1 \right\} \\
&= \left\{ (s, s') \;\middle|\; i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\} \\
&= W.
\end{aligned}$$

Hence W is range inclusive. Let V be the relation defined as

$$\begin{aligned}
&I(S - rng(W)) \circ W \\
&= \left\{ (s, s') \;\middle|\; i(s) \neq n+1 \wedge i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}.
\end{aligned}$$

We take V' as

$$V' = \left\{ (s, s') \;\middle|\; i(s) \neq n+1 \wedge a(s') = a(s) \wedge i(s') = n+1 \right.$$
$$\left. \wedge\, x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}.$$

Whence we get

$$W' = V' \cup I(rng(W))$$
$$= \left\{ (s, s') \;\middle|\; i(s) \neq n+1 \wedge a(s') = a(s) \wedge i(s') = n+1 \right.$$
$$\left. \wedge\, x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}$$
$$\cup \{(s, s') \mid i(s) = n+1 \wedge a(s') = a(s) \wedge i(s') = i(s) \wedge x(s') = x(s)\}$$
$$= \left\{ (s, s') \;\middle|\; i(s) \neq n+1 \wedge a(s') = a(s) \wedge i(s') = n+1 \right.$$
$$\left. \wedge\, x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}$$
$$\cup \left\{ (s, s') \;\middle|\; i(s) = n+1 \wedge a(s') = a(s) \wedge i(s') = n+1 \right.$$
$$\left. \wedge\, x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}$$
$$= \left\{ (s, s') \;\middle|\; a(s') = a(s) \wedge i(s') = n+1 \wedge x(s') = x(s) + \sum_{k=i(s)}^{n} a(s)[k] \right\}.$$

This relation W' is total, deterministic, and range identical. □

4.3. *Heuristic G2: making a relation transitive*

If we place heuristic $G1$ upstream of heuristic $T1$, the conditions that we must now realize are:

- for $G1$:
  - W is total,
  - W is range inclusive.
- for $T2$:
  - W is total,
  - W is range inclusive,
  - $\kappa(W) \circ I(rng(W))$ is total.

We focus our attention now on the third condition of heuristic $T2$, namely that $\kappa(W) \circ I(rng(W))$ is total. Rather than to realize this condition, we propose to realize a stronger condition, namely, that $\kappa(W) \circ I(rng(W))$ equals W. Because the latter is total, so is the former.

Lemma 1 (Heuristic $G2$). *Let W be a total, transitive relation which is range inclusive. Then*

$$\kappa(W) \circ I(rng(W)) = W.$$

Proof. We have proven in the lemma to heuristic $T2$ that if W is range inclusive then $\kappa(W) \circ I(rng(W)) \subseteq W$. It suffices to prove here that if W is transitive then $W \subseteq \kappa(W) \circ I(rng(W))$.

$$\begin{aligned}
W &= \{(s, s') \mid s' \in s.W \wedge s' \in rng(W)\} \\
&\qquad \text{paraphrase of } (s, s') \in W \\
&\subseteq \{(s, s') \mid s'.W \subseteq (s.W).W \wedge s' \in rng(W)\} \\
&\qquad \text{by monotony of image sets} \\
&= \{(s, s') \mid s'.W \subseteq s.(W \circ W) \wedge s' \in rng(W)\} \\
&\qquad \text{by identity} \\
&\subseteq \{(s, s') \mid s'.W \subseteq s.W \wedge s' \in rng(W)\} \\
&\qquad \text{by transitivity of } W. \\
&= \{(s, s') \mid \emptyset \neq s'.W \subseteq s.W \wedge s' \in rng(W)\} \\
&\qquad \text{by totality of } W. \\
&= \kappa(W) \circ I(rng(W)) \\
&\qquad \text{by definition of } \kappa() \text{ and } I().
\end{aligned}$$

□

This proposition provides that the third condition of heuristic $T2$ holds whenever relation W is range inclusive and transitive. We concentrate on ways to ensure the transitivity of a relation by generalization.

Lemma 2 (Heuristic $G2$). *Let W be a total relation which is range inclusive. Then*

$$W' = I(S - rng(W)) \circ W \cup I(rng(W))$$

is transitive and is more-defined than W.

Proof. Let I' be $I(rng(W))$ and I'' be $I(S - rng(W))$. We compute $W' \circ W'$, and show it to be a subset of W'.

$$\begin{aligned}
W' \circ W' &= (I'' \circ W \cup I') \circ (I'' \circ W \cup I') \\
&\qquad \text{by the proposed formula} \\
&= I'' \circ W \circ I'' \circ W \cup I'' \circ W \circ I' \cup I' \circ I'' \circ W \cup I' \circ I' \\
&\qquad \text{by distributivity} \\
&= I'' \circ W \circ I' \cup I' \\
&\qquad \text{since } W \circ I'' \text{ and } I' \circ I'' \text{ are empty, and } I' \circ I' = I' \\
&= I'' \circ W \cup I' \qquad \text{since } W \circ I' = W \\
&= W' \qquad \text{by definition.}
\end{aligned}$$

To prove that W' is more-defined than W, it is sufficient to note that they both have the same domain, and that the former is a subset of the latter. □

From these two lemmas, we derive the following heuristic.

Heuristic *G2*. Given a total relation W which is range inclusive, take

$$W' = I(S - rng(W)) \circ W \cup I(rng(W)).$$

The two key properties of this heuristic are:

- W' is more-defined than W,
- W' verifies the third condition of heuristic $T2$.

The first premise provides that $G2$ is valid; the second provides that $G2$ is useful.

Example. Let S and W be defined as follows.

$$S = \{0, 1, 2\},$$

$$W = \{(0, 0), (0, 1), (1, 0), (1, 1), (2, 0)\}.$$

Then

$$0.W = \{0, 1\}, \qquad 1.W = \{0, 1\}, \qquad 2.W = \{0\}.$$

Hence

$$\kappa(W) = \{(0, 0), (1, 1), (2, 2), (0, 1), (1, 0), (0, 2), (1, 2)\}$$

and

$$\kappa(W) \circ I(rng(W)) = \{(0, 0), (1, 1), (1, 0), (0, 1)\}.$$

thus,

$$dom(W) \neq dom(\kappa(W) \circ I(rng(W))).$$

We apply heuristic $G2$ to W. First, we check whether W verifies the conditions of applicability of heuristic $G2$.

Totality. Clearly, W is total.

Range inclusiveness. $W \cap I = \{(0, 0), (1, 1)\}$. Hence,

$$W \circ (W \cap I) = \{(0, 0), (0, 1), (1, 0), (1, 1), (2, 0)\} \circ \{(0, 0), (1, 1)\}$$

$$= \{(0, 0), (0, 1), (1, 0), (1, 1), (2, 0)\} = W.$$

By $G2$, we get

$$W' = \{(2, 0), (1, 1), (0, 0)\}.$$

For the sake of illustration, we check that W' does indeed satisfy the desired condition (namely, that $\kappa(W') \circ I(rng(W'))$ is total). First, we compute the kernel of W'.

$$0.W' = \{0\}, \qquad 1.W' = \{1\}, \qquad 2.W' = \{0\}.$$

Hence

$$\kappa(W') = \{(0, 0), (1, 1), (2, 2), (0, 2), (2, 0)\}.$$

Whence we deduce

$$\kappa(W') \circ I(rng(W')) = \{(0, 0), (1, 1), (2, 0)\},$$

and

$$S = dom(\kappa(W') \circ I(rng(W'))). \qquad \square$$

4.4. Heuristic G3: making a relation range inclusive

If we place heuristic $G2$ upstream of heuristic $T2$, and consider that $G1$ is upstream of $T1$, we now have the following conditions to realize:

- for $G1$:
  - W is total,
  - W is range inclusive.
- for $G2$:
  - W is total,
  - W is range inclusive.

Interestingly, $G1$ and $G2$ have the same list of applicability conditions. We focus our attention on the second element of this list. We have the following lemma.

Lemma (Heuristic $G3$). *Let W be a total relation which is range generative. Then $W' = W \circ (W \cap I)$ is more-defined than W. Furthermore, it is total, and range inclusive.*

Proof. Because W is range generative, W and W' have the same domain. By construction, W is a subset of W' (as $W \cap I \subseteq I$). Hence W' is more-defined than W. W' is total because it has the same domain as W, which is total. To conclude this proof, we show that W' is range inclusive. Let s be an element of $rng(W')$; then s is an element of $rng(W \cap I)$; then (s, s) is an element of $(W \cap I)$; then (s, s) is an element of $W \circ (W \cap I) = W'$. $\square$

Whence the following heuristic.

Heuristic *G3*. Given a total relation W which is range generative, take

$$W' = W \circ (W \cap I).$$

The two main features of this heuristic are the following:

- W' is more-defined than W,
- W' is range inclusive.

The first premise ensures the validity of this heuristic, while the second premise ensures its usefulness. Furthermore, it is worthwhile to notice that whenever W is 3-iterative, i.e. total and range inclusive, heuristic $G3$ maps it into itself (a pleasant sign of minimality).

Example. We let W be the following relation on the space S defined by the declaration

$$a, b, c: \textbf{natural},$$

$$W = \{(s, s') \mid a(s') = a(s) + b(s)\}.$$

Relation W is clearly total; it is easy to see that it is not range inclusive, as $a(s') = a(s) + b(s)$ is not a logical consequence of $s' = s$. In order to apply heuristic $G3$, we check that it is range generative. To this effect, we compute

$$\begin{aligned} W \circ (W \cap I) &= \{(s, s') \mid a(s') = a(s) + b(s)\} \circ \{(s, s') \mid a(s') = a(s) + b(s) \wedge s' = s\} \\ &= \{(s, s') \mid a(s') = a(s) + b(s) \wedge a(s') = a(s') + b(s')\} \\ &= \{(s, s') \mid a(s') = a(s) + b(s) \wedge b(s') = 0\}. \end{aligned}$$

Hence,

$$dom(W \circ (W \cap I)) = S = dom(W).$$

Hence W is in second iterative form. We compute

$$W' = \{(s, s') \mid a(s') = a(s) + b(s) \wedge b(s') = 0\}.$$

We leave it to the reader to check that W' is range inclusive. □

4.5. Heuristic G4: making a specification range generative

If we place heuristic $G3$ upstream of heuristics $G1$ and $G2$, we now get the following conditions.

- for $G3$:
  - W is total,
  - W is range generative.

We focus our attention on the first condition of this heuristic. We have the following lemma.

Lemma (Heuristic $G4$). *Let W be a relation on S which is iterative. Then*

$$W' = W \cup I(S - dom(W)) \circ L$$

is more-defined than W. Furthermore, W' is total and range generative.

Proof. Relation W' has a larger domain than relation W; furthermore, on the domain of W, $s.W' = s.W$. Hence W' is more-defined than W. Clearly, W' is total. We prove below that it is range generative; because the case $dom(W) = S$ is trivial, we assume that $dom(W)$ is a proper subset of S. To prove that W' is range generative, we compute

$$
\begin{aligned}
&W' \circ (W' \cap I) \\
&\quad = (W \cup I(S - dom(W)) \circ L) \circ ((W \cup I(S - dom(W)) \circ L) \cap I) \\
&\qquad \text{by definition of } W' \\
&\quad = (W \cup I(S - dom(W)) \circ L) \circ (W \cap I \cup (I(S - dom(W)) \circ L) \cap I) \\
&\qquad \text{distributing } \cap \text{ over } \cup \\
&\quad = (W \cup I(S - dom(W)) \circ L) \circ (W \cap I \cup I(S - dom(W))) \\
&\qquad \text{because } I(S - dom(W)) \circ L \cap I = I(S - dom(W)) \\
&\quad = W \circ (W \cap I) \cup W \circ I(S - dom(W)) \cup M \circ (W \cap I) \\
&\qquad \cup M \circ I(S - dom(W)) \\
&\qquad \text{distributing the union over the relative product, and letting } M \text{ be} \\
&\qquad I(S - dom(W)) \circ L.
\end{aligned}
$$

Now, we compute (algebraically) the domain of this expression, and we distribute it over the union. This yields

$$
\begin{aligned}
&W' \circ (W' \cap I) \circ L \\
&\quad = W \circ (W \cap I) \circ L \cup W \circ I(S - dom(W)) \circ L \cup M \circ (W \cap I) \circ L \\
&\qquad \cup M \circ I(S - dom(W)) \circ L \\
&\qquad\quad \text{by the above development} \\
&\quad = W \circ (W \cap I) \circ L \cup W \circ I(S - dom(W)) \circ L \cup M \circ (W \cap I) \circ L \\
&\qquad \cup I(S - dom(W)) \circ L \circ I(S - dom(W)) \circ L \\
&\qquad\quad \text{by the formula of } M \\
&\quad = W \circ (W \cap I) \circ L \cup W \circ I(S - dom(W)) \circ L \cup M \circ (W \cap I) \circ L \\
&\qquad \cup I(S - dom(W)) \circ L \\
&\qquad\quad \text{because } L \circ I(S - dom(W)) \circ L = L, \text{ due to the hypothesis} \\
&\qquad\quad I(S - dom(W)) \neq \emptyset \\
&\quad = W \circ (W \cap I) \circ L \cup W \circ I(S - dom(W)) \circ L \cup M \circ (W \cap I) \circ L \cup M \circ L \\
&\qquad\quad \text{by definition of } M, \text{ and because } L = L \circ L \\
&\quad = W \circ (W \cap I) \circ L \cup W \circ I(S - dom(W)) \circ L \cup M \circ L \\
&\qquad\quad \text{because } W \cap I \subseteq I, \text{ hence the third term is a subset of the fourth.}
\end{aligned}
$$

In order to prove that W' is range generative, we must prove that this expression

equals L (since W' is total). We proceed by equivalences;

$$W \circ (W \cap I) \circ L \cup W \circ I(S - dom(W)) \circ L \cup M \circ L = L$$
$$\Leftrightarrow$$
$$W \circ (W \cap I) \circ L \cup W \circ I(S - dom(W)) \circ L = L - M \circ L$$

because $M \circ L \cap (W \circ (W \cap I) \circ L \cup W \circ I(S - dom(W)) \circ L) = \emptyset$, and by the identity that if $A \cap B = \emptyset$ then $A \cup B = S$ is equivalent to $A = S - B$.

$$\Leftrightarrow$$
$$W \circ (W \cap I) \circ L \cup W \circ I(S - dom(W)) \circ L = W \circ L,$$

by definition of M.

Because this last condition is a hypothesis, we establish the desired conclusion: W' is range generative. □

Heuristic *G*4. Given a relation W which is iterative, take

$$W' = W \cup I(S - dom(W)) \circ L.$$

The two main features of this heuristic are the following:

- W' is more-defined than W,
- W' is total and range generative (if W is in first iterative form).

The first premise ensures the validity of this heuristic (as an instance of the generalization rule) while the second premise ensures its usefulness (in realizing a stronger condition). We leave it to the reader to convince himself that if W is total and in first iterative form then heuristic $G4$ maps it into itself (a desirable minimality property). Notice also that this heuristic maps W into the least defined relation W' that is total and more-defined than W.

Example. We let W be the following relation on space S defined by the declaration a, b, c: **integer**

$$W = \{(s, s') \mid b(s) \geq 0 \wedge a(s') = a(s) + b(s)\}.$$

We check whether W is iterative.

$$W \circ (W \cap I) = \{(s, s') \mid b(s) \geq 0 \wedge a(s') = a(s) + b(s)\}$$
$$\circ \{(s, s') \mid b(s) \geq 0 \wedge a(s) = a(s) + b(s) \wedge s' = s\}$$
$$= \{(s, s') \mid b(s) \geq 0 \wedge a(s') = a(s) + b(s) \wedge b(s') = 0\}.$$

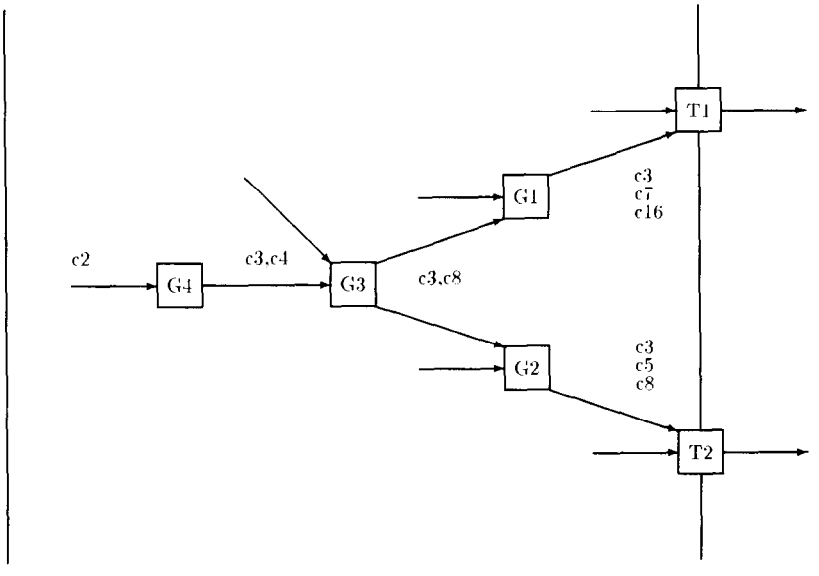
Because the domain of this term is already equal to $dom(W)$, we need not even compute the term $(W \circ I(S - dom(W)))$. We can conclude that W is in first iterative form. Heuristic $G4$ maps specification W into:

$$W' = \{(s, s') \mid b(s) \geq 0 \wedge a(s') = a(s) + b(s)\} \cup \{(s, s') \mid b(s) < 0\}$$
$$= \{(s, s') \mid b(s) \geq 0 \Rightarrow a(s') = a(s) + b(s)\}. \quad \square$$

If we place heuristic $G4$ upstream of heuristic $G3$, we now have the condition:

- for $G4$:
  - W is iterative.

We have in effect established, constructively, that any specification that meets this condition can be processed by the network of heuristics, to produce a solution under the form of a loop body specification. Any specification that meets this condition is processed by $G4$, then $G3$, then, depending on whether we wish to preserve non-determinacy (to leave future design options open) or to get rid of it (trade flexibility for simplicity), we apply the sequence $G2/T2$ or the sequence $G1/T1$. Note that it is likely (and, by experience, common) that upon exiting from a heuristic,



c2: W iterative
c3: W total
c4: W range generative
c5: $\kappa(W) * I(rng(W))$ total
c8: W range inclusive
c7: W deterministic
c16: W range identical

Fig. 1. The network of generalization/iteration heuristics.

a specification has a stronger property than the property that the heuristic is designed to realize. Hence it may be useful to consider skipping heuristics, even though the network shows them in strict sequence. The whole generalization/iteration network is represented in Fig. 1.

5. Completeness of the network

In the previous section we have derived the condition under which a specification can be processed by the network of generalization/iteration heuristics; this we call the *applicability* condition. On the other hand, we define the *feasibility* condition, as the condition under which a specification W admits a solution under the form of a while statement. By construction, the applicability condition logically implies the feasibility condition. In order to assess the completeness of our network of heuristics, we must discuss whether the feasibility condition logically implies the applicability condition. This is the subject of this section. We have the proposition.

Proposition (Completeness of the Network). *If specification W has a solution under the form*

(**while** t **do** b),

then W is iterative.

Proof. Let W be a relation that has a solution under the form

(**while** t **do** b).

By the proposition of the iteration rule (Section 2), there exists a relation W' which is total, range identical, and is more-defined than W. We assume that W is not iterative, and show that this leads us to a contradiction.

Because the formula

$$dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W))) \subseteq dom(W)$$

is a tautology, the applicability condition can be written as:

$$dom(W) \subseteq dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W))).$$

The negation of this condition can be written as:

$$\exists s\colon s \in dom(W) \wedge s \notin (dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W)))).$$

Because W' is more-defined than W, we deduce from $s \in dom(W)$ that $s \in dom(W') \wedge s.W' \subseteq s.W$, by which we replace it.

$$\Rightarrow \exists s\colon s \in dom(W') \wedge s.W' \subseteq s.W \wedge s \notin (dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W)))).$$

From $s \in dom(W')$, there exists t such that $(s, t) \in W'$.

$$\Rightarrow \exists s, t: (s, t) \in W' \wedge s.W' \subseteq s.W \wedge s \notin (dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W)))).$$

From $(s, t) \in W' \wedge s.W' \subseteq s.W$, we deduce $(s, t) \in W' \wedge (s, t) \in W$.

$$\Rightarrow \exists s, t: (s, t) \in W' \wedge (s, t) \in W \wedge s \notin (dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W)))).$$

Because W' is range identical, $(s, t) \in W'$ implies $t.W' = \{t\}$.

$$\Rightarrow \exists s, t: (s, t) \in W \wedge t.W' = \{t\} \wedge s \notin (dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W)))).$$

From here on, we proceed by case analysis, on whether t is or is not in $dom(W)$.

Case 1: $t \in dom(W)$. Because W' is more-defined than W, we deduce from $t \in dom(W)$ that $t \in dom(W')$ and $t.W' \subseteq t.W$; because $t.W' = \{t\}$, we get $(t, t) \in W$.

$$\Rightarrow \exists s, t: (s, t) \in W \wedge (t, t) \in W \wedge s \notin (dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W)))).$$

Because (t, t) is also an element of I, we get

$$\Rightarrow \exists s, t: (s, t) \in W \wedge (t, t) \in W \cap I \wedge s \notin (dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W)))).$$

From $(s, t) \in W \wedge (t, t) \in W \cap I$, we deduce by the definition of relative product that $(s, t) \in W \circ (W \cap I)$, whence $s \in dom(W \circ (W \cap I))$.

$$\Rightarrow \exists s, t: s \in dom(W \circ (W \cap I)) \wedge s \notin (dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W)))).$$

This is clearly a contradiction.

$$\Rightarrow \textbf{false}.$$

Case 2: $t \notin dom(W)$. We consider again the condition as it was before we have taken the hypothesis of case 1:

$$\exists s, t: (s, t) \in W \wedge t.W' = \{t\} \wedge s \notin (dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W)))).$$

From $(s, t) \in W \wedge t \notin dom(W)$ we deduce $(s, t) \in W \circ I(S - dom(W))$. Hence $s \in dom(W \circ I(S - dom(W)))$.

$$\Rightarrow \exists s, t: s \in dom(W \circ I(S - dom(W))) \wedge s \notin (dom(W \circ (W \cap I)) \cup dom(W \circ I(S - dom(W)))).$$

This is clearly a contradiction.

$$\Rightarrow \textbf{false}.$$

We have assumed that W has a while statement as solution but does not verify the applicability condition of the network of heuristics, and have found that our assumption leads to a contradiction. Hence whenever a specification has a solution as a while statement, it meets the applicability condition. □

By virtue of this proposition we claim that the network of heuristics that we have constructed is complete, in the sense that it is capable of handling all the specifications that admit a decomposition of the desired form.

6. An illustrative example

In this section we illustrate the network of heuristics that we have presented in this paper on a non trivial example; what is non trivial about the example we have chosen is its data structures, rather than the complexity of the function it computes. The point we wish to make in this section is that the mathematics that we have presented in this paper is applicable, not only to trivial data structures (as the examples dealt with so far may lead to believe), but also to arbitrarily compound data structures, provided they are abstracted properly, at a meaningful level.

Before we proceed with the example, we present a mathematical notion that we need for our developments: the *conjugate kernel*, an operator that performs a division of a relation by another.

6.1. Conjugate kernels

The *conjugate kernel* (see [9] for more detail) of relation R by relation Q is the relation denoted by $\kappa(R, Q)$ and defined by:

$$\kappa(R, Q) = \{(s, s') \mid \emptyset \neq s'.Q \subset s.R\}.$$

It stems from this definition that the kernel of a relation R (see Section 3.1) is nothing but the conjugate kernel of R by itself.

The *weakest prespecification problem* consists of determining the least defined relation X that satisfies simultaneously the following two equations:

$$XQ = R,$$

$$X \subseteq L\hat{Q}.$$

The first equation expresses that the pair (X, Q) defines a sequence decomposition of relation R, and the second equation expresses that the range of X is a subset of the domain of Q.

We have found in [9] that whenever it is feasible, the conjugate kernel of R and Q is an optimal solution to the above problem. Also, we have found that whenever Q is deterministic and has a range that is larger than the range of R, the conjugate kernel of R by Q is nothing but $R\hat{Q}$. We will use conjugate kernels for a stepwise development of sequential programs: given specification R, we *divide* it little by little using the conjugate kernel, until we find a relation which is simple enough to be implemented without further decomposition; or until we obtain a relation which is reflexive,[3] and which can be implemented by an empty program.

6.2. Counting the nodes of a tree

We consider the specification of a while loop to count the number of nodes in a binary tree; several proofs of correctness of this problem are studied by [19]. The space that we define for this specification is,

tree: treetype;

stack: stack-of-tree-type;

number: **integer**,

where we assume further that the stack does not contain empty trees. We let W be the following relation

$$W = \left\{ (s, s') \;\middle|\; number(s') = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t) \right\},$$

where **nodes**(t) is the number of nodes of tree t. One may wonder why the specification of W is so defined, when the purpose of this program is to compute the number of nodes in a tree. This has to do with the difference between the function of an *initialized* while loop and the function of an *uninitialized* while loop. An initialized while loop would place the tree in the stack, then invoke the while loop to compute the number of nodes in the single tree which is in the stack. But the while loop would in fact compute the number of nodes of any number of trees one may have pushed on the stack, because it iterates as long as the stack is not empty.

[3] It suffices that the resulting quotient be reflexive on its domain, rather than on all of S.

First, we check the iterative properties of this relation. Because it is total, we look for at least the range generative property; because this relation is surjective ($I(rng(\,)) = I$) but not reflexive ($I \not\subseteq W$), we look for at most the range generative property. We compute,

$$
\begin{aligned}
&W \circ (W \cap I) \\
&= \left\{ (s, s') \;\middle|\; number(s') = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t) \right\} \\
&\quad \circ \left\{ (s, s') \;\middle|\; number(s) = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t) \wedge s' = s \right\} \\
&= \left\{ (s, s') \;\middle|\; number(s') = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t) \right\} \\
&\quad \circ \left\{ (s', s') \;\middle|\; \sum_{t \in stack(s')} \mathbf{nodes}(t) = 0 \right\}
\end{aligned}
$$

by arithmetic

$$
\begin{aligned}
&= \left\{ (s, s') \;\middle|\; number(s') = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t) \right. \\
&\qquad \left. \wedge\, \mathbf{empty}(stack(s')) \right\}
\end{aligned}
$$

because the trees in the stack cannot be empty.

This relation has the same domain as W, they are both total. Hence W is range generative. Because it is total, it is 2-iterative. We apply heuristic $G3$, yielding:

$$
\begin{aligned}
W' = \left\{ (s, s') \;\middle|\; number(s') = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t) \right. \\
\left. \wedge\, \mathbf{empty}(stack(s')) \right\}.
\end{aligned}
$$

This specification is, by construction, total and range inclusive. We may apply the sequence $G1/T1$ or the sequence $G2/T2$; for the purposes of this example, we favor simplicity over non-determinacy. Hence we select the former course of action. Application of $G1$ proceeds as follows:

$$
\begin{aligned}
&V = I(S - rng(W')) \circ W' \\
&= \left\{ (s, s') \;\middle|\; \neg \mathbf{empty}(stack(s)) \right. \\
&\qquad \left. \wedge\, number(s') = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t) \wedge \mathbf{empty}(stack(s')) \right\}.
\end{aligned}
$$

We must take a subset of V, say V', such that V' is deterministic and has the same domain as V; to take a subset of V it suffices to add conjuncts to the description

of V, so as to determine tree(s'); for the sake of uniformity with the second component of W'', which is $I(rng(W))$, we take $tree(s') = tree(s)$. This yields,

$$W'' = \left\{ (s, s') \;\middle|\; tree(s') = tree(s) \right.$$
$$\left. \wedge\ number(s') = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t) \wedge \mathbf{empty}(stack(s')) \right\}.$$

Specification W'' is, by construction, deterministic, total, and range identical. We apply heuristic $T1$, yielding the following development. First, we need to chose a well-founded ordering that contains $I(S - rng(W''))\circ W''$. We compute,

$$I(S - rng(W''))\circ W''$$
$$= \left\{ (s, s') \;\middle|\; \neg\mathbf{empty}(stack(s)) \wedge tree(s') = tree(s) \right.$$
$$\left. \wedge\ number(s') = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t) \wedge \mathbf{empty}(stack(s')) \right\}$$
$$= \left\{ (s, s') \;\middle|\; \neg\mathbf{empty}(stack(s)) \wedge \sum_{t \in stack(s)} \mathrm{nodes}(t) > 0 \wedge tree(s') = tree(s) \right.$$
$$\wedge\ number(s') = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t) \wedge \mathbf{empty}(stack(s'))$$
$$\left. \wedge \sum_{t \in stack(s')} \mathbf{nodes}(t) = 0 \right\}$$
$$\subseteq \left\{ (s, s') \;\middle|\; \sum_{t \in stack(s)} \mathbf{nodes}(t) > 0 \wedge \sum_{t \in stack(s')} \mathbf{nodes}(s') = 0 \right\}$$
$$\subseteq \left\{ (s, s') \;\middle|\; \sum_{t \in stack(s)} \mathbf{nodes}(t) > \sum_{t \in stack(s')} \mathbf{nodes}(t) \right\}.$$

This relation, which is well-founded, is a superset of $I(S - rng(W''))\circ W''$; we take it to be our relation GT. Whence application of heuristic $T1$ yields the following loop body specification:

$$B = \left\{ (s, s') \;\middle|\; \neg\mathbf{empty}(stack(s)) \right.$$
$$\wedge\ number(s') + \sum_{t \in stack(s')} \mathbf{nodes}(t) = number(s) + \sum_{t \in stack(s)} \mathbf{nodes}(t)$$
$$\left. \wedge\ tree(s') = tree(s) \wedge \sum_{t \in stack(s')} \mathbf{nodes}(t) < \sum_{t \in stack(s)} \mathbf{nodes}(t) \right\}.$$

Now we can implement B by a sequence of statements, using the conjugate kernel operation. First, we rewrite B as follows:

$$B = \left\{ (s, s') \;\middle|\; \neg\textbf{empty}(stack(s)) \right.$$
$$\wedge\ number(s') + \sum_{t \in stack(s')} \textbf{nodes}(t) = number(s) + \sum_{t \in stack(s)} \textbf{nodes}(t)$$
$$\left. \wedge\ tree(s') = tree(s) \wedge number(s') > number(s) \right\}.$$

In order to satisfy the last conjunct of this relation, we may decide to increase the variable *number* by 1; according to [9], the specification that must be satisfied before *number* is incremented is $\kappa(B, Q)$, where

$$Q = \{(s, s') \mid tree(s') = tree(s) \wedge number(s') = number(s) + 1$$
$$\wedge\ stack(s') = stack(s)\}.$$

The developments given in [9] provide that, because Q is deterministic, $\kappa(B, Q) = B\hat{Q}$. We find,

$$B' = B\hat{Q}$$
$$= \left\{ (s, s') \;\middle|\; tree(s') = tree(s) \right.$$
$$\wedge\ number(s') + 1 + \sum_{t \in stack(s')} \textbf{nodes}(t) = number(s) + \sum_{t \in stack(s)} \textbf{nodes}(t)$$
$$\left. \wedge\ number(s') \geqslant number(s) \right\}.$$

One way to satisfy this specification is to decrease the number of nodes in the trees stored in *stack* by 1. To do so, we "divide" relation B' by relation Q', which is defined as follows:

$$Q' = \left\{ (s, s') \;\middle|\; tree(s') = tree(s) \wedge \sum_{t \in stack(s')} \textbf{nodes}(t) = \sum_{t \in stack(s)} \textbf{nodes}(t) - 1 \right.$$
$$\left. \wedge\ number(s') = number(s) \right\}.$$

The division yields,

$$B'' = \kappa(B', Q')$$
$$= \left\{ (s, s') \;\middle|\; tree(s') = tree(s) \right.$$
$$\wedge\ number(s') + \sum_{t \in stack(s')} \textbf{nodes}(t) = number(s) + \sum_{t \in stack(s)} \textbf{nodes}(t)$$
$$\left. \wedge\ number(s') \geqslant number(s) \right\}.$$

This relation is reflexive, hence the division stops here. We can decompose relation Q' by the union rule: we rewrite Q' as the union of $Q'_1 \cup Q'_2$, with

$$Q'_1 = \left\{ (s, s') \;\middle|\; \mathbf{leaf}(\mathbf{top}(stack(s))) \wedge tree(s') = tree(s) \right.$$
$$\left. \wedge \sum_{t \in stack(s')} \mathbf{nodes}(t) = \sum_{t \in stack(s)} \mathbf{nodes}(t) - 1 \wedge number(s') = number(s) \right\}$$

and

$$Q'_2 = \left\{ (s, s') \;\middle|\; \neg\mathbf{leaf}(\mathbf{top}(stack(t))) \wedge tree(s) = tree(s') \right.$$
$$\left. \wedge \sum_{t \in stack(s')} \mathbf{nodes}(t) = \sum_{t \in stack(s)} \mathbf{nodes}(t) - 1 \wedge number(s') = number(s) \right\}.$$

Relation Q'_1 can be satisfied by **pop**ing the top of the stack, while relation Q'_2 can be satisfied by **pop**ing the top of the stack then **push**ing its non-empty sons. □

7. Conclusion

7.1. Summary

In this paper we have attempted to analyze the mathematical mechanisms that underly the construction of iterative algorithms from relational specifications. We have assumed the data structures to be predefined, and properly axiomatized, and have concentrated our attention on the stepwise development of the iterative algorithms that manipulate these data structures.

We have recognized that the development of an (non initialized) iterative program from a relational specifications proceeds in two steps: a *generalization* step, during which the specification is made sufficiently general to undergo the iterative decomposition; a *decomposition* step, during which a loop body specification is derived from the specification of the whole while loop.[4] Also, we have proposed systematic *heuristics* for carrying out these two steps: each heuristic recognizes specific properties of the specification at hand, then proposes a solution in the form of a transformed (generalized or decomposed) specification.

For each one of these heuristics, we have endeavored to analyze the mathematical processes that come into play in deriving the constructed specification from the given specification. Specifically, we are interested in analyzing what parameters appear in the formulation of the constructed specification, and in determining, among these parameters, which are derived constructively from the given specification, and which are left to the discretion of the programmer. Also, among the latter, we are interested in investigating the limits within which these parameters

[4] Not incidentally, this two phase process has similarities with the process of generalizing an assertion, then proving it by induction.

can be determined. We have found that the decomposition heuristics, $T1$ and $T2$, derive the loop body specification B from the given specification W in a constructive, formula based manner, leaving a single parameter to the discretion of the programmer. This parameter, a well founded ordering, must be chosen to be a superset of some relation; our experience shows that this relation typically contains a fairly visible well founded ordering, whose identification is made all the more easier. As for the generalization heuristics, we have found three of them to be totally constructive, namely $G2$, $G3$ and $G4$. As for $G1$, it leaves some latitude to the programmer; as the example in Section 6 shows, this latitude can be used to simplify the expression of the constructed specification.

7.2. Relationship to other works

Our work shares its general objective with a number of other works, including (among recent references): Gries' *The Science of Programming* [11], Backhouse's *Program Construction and Verification* [1], Hehner's *The Logic of Programming* [12], Jones' *Systematic Software Development* [16], Dijkstra and Feijen's *A Method of Programming* [10], and Morgan's *Constructing Programs from Specifications* [24]. Our work can be identified with the following premises:

- *Its Focus.* This work focuses on algorithmic refinement, rather than data refinement. The spaces that we consider are not changed throughout the construction process, and are assumed to be properly axiomatized, at a meaningful level of abstraction.
- *Its level of Abstraction.* We deal with software components at the *program* level of abstraction; hence we only handle traditional programming language constructs, and do not deal with such matters as intermodule control.
- *Its Mathematical Tool.* Specifications are represented by relations and program construction is performed by stepwise transformation of specifications. A Tarski-like algebra of relations proves effective in capturing these problems and solving them.

Our work can be distinguished from works such as Gries' [11] and Dijkstra's [10] on the basis of two features: first, its tool (relations, versus precondition/postcondition pairs); second, its emphasis (understanding more of the mechanics of program construction, versus putting our current understanding to work on ever more complex examples). The second distinguishing feature is more significant than the first. Because of its emphasis on *deriving programs by computation rather than by inspection* the work of Billington et al. [3] is closer to our work, although it differs by its tool (relational algebra, versus logic); this is a fairly minor difference, we believe. The work on weakest prespecifications of Hoare and He [13] uses the same kind of relational algebra as we do, but with different hypotheses (totality of specifications, hence the use of the inclusion relationship rather than the *more-defined* relationship). The work of Backhouse et al. [2] also uses relational algebra, but concentrates on data refinement rather than algorithmic refinement. The influence of the work of

Mills et al. [18, 23] on our work is visible; we consider our work to be based on several of Mill's basic premises.

7.3. Perspectives

While it has a number of theoretical properties (such as correctness preservation, minimality of some heuristics, completeness, ...) the network of heuristics that we have presented in this paper poses some difficulties in practice. Most of these difficulties come from a single feature: All the formulae given in the heuristics are representation blind, i.e. do not take into account how easy or how difficult it is to represent the specifications at hand. Improvements of the network in light of this remark are currently under investigation. Also under investigation is the capability to add variable and data structures on the fly, as the algorithm is being developed. In theory, variable introduction poses no difficulty, since it is merely a cartesian product operation; integrating it in the construction process, among the heuristics, and making it as imperative as our current heuristics are, may be more of a challenge.

On a more practical side, many of the computations that are involved in these heuristics can be automated - and should be, if these results are to be useful to the practicing programmer. This is currently being investigated. We consider that the aggregate of heuristics we have presented here, as well as others we have developed, can conceivably be used as the blueprint for a genuine automatic programmer. This programmer would be genuinely automatic, in the sense that it generates original algorithms, rather than to index a library of existing algorithms or algorithm patterns.

Acknowledgements

This research results from research and teaching we have carried out recently at various institutions, including: Laval University, the University of Tunis, Oakland University, University of Klagenfurt, University of Queensland. We are grateful to these institutions for the opportunity. We are also grateful to Dr Jules Desharnais, from Laval University, for his continuous feedback and insights.

References

[1] R.C. Backhouse, *Program Construction and Verification* (Prentice-Hall, Englewood Cliffs, NJ, 1986).

[2] R. Backhouse, P. De Bruin, G. Malcolm, E. Voermans and J. Van der Woude, A relational theory of datatype, *Workshop on Constructive Algorithms: the Role of Relations in Program Development*, Hollum-Ameland, Netherlands, September 1990.

[3] D. Billington, D.E. Abel, T.A. Chorvat, R.G. Dromey, D.D. Grant and F. Suraweera. *Program Derivation: A Clarification of Some Issues*, Programming Methodology Research Group, Griffith University, Brisbane, Australia, 1989.

[4] J.W. de Bakker and D. Scott, A theory of programs, in: *J.W. de Bakker, 25 Jaar Semantiek, Liber Amicorum* (CWI, Amsterdam, 1989) 1-30.

[5] J.W. de Bakker and W.P. de Roever, A calculus for recursive program schemes, *Automata, Languages and Programming, Proceedings of a symposium organized by IRIA*, Rocquencourt, France, July 1972 (North-Holland, Amsterdam, 1973) 167–196.
[6] R. Berghammer, G. Schmidt and H. Zierer, *Symmetric Quotients*, Technical Report, TUM-I8620, Technische Universität München, 1986.
[7] R. Berghammer, G. Schmidt and H. Zierer, Symmetric quotients and domain constructions, *Inform. Proc. Lett.* **33** (1989) 163–168.
[8] J. Desharnais, *Abstract Relational Semantics*, PhD dissertation, Dept. of Computer Science, McGill University, Montreal, Canada, 1989.
[9] J. Desharnais, A. Jaoua, F. Mili, N. Boudriga and A. Mili, The conjugate kernel: an operator for program construction, *Workshop on Constructive Algorithms: the Role of Relations in Program Development.* Hollum-Ameland, Netherlands, September 1990.
[10] E.W. Dijkstra and W.H.J. Feijen, *A Method of Programming* (Addison-Wesley, Reading, MA, 1988).
[11] D. Gries, *The Science of Programming* (Springer, New York, 1981).
[12] E.C.R. Hehner, *The Logic of Programming* (Prentice-Hall Intl., London, 1986).
[13] C.A.R. Hoare and J. He, The weakest prespecification, *Fundam. Inform.* **9** (1986), Part I: 51–84, Part II: 217–252.
[14] C.A.R. Hoare and J. He, The weakest prespecification, *Inform. Process. Lett.* **24** (1987) 127–132.
[15] A. Jaoua, N. Boudriga, J-L. Durieux and A. Mili, Pseudo-invertibility, a measure of regularity of relations, *Theoret. Comput. Sci.* **79**(2) (1991) 323–339.
[16] C.B. Jones, *Systematic Software Development Using VDM* (Prentice-Hall, Englewood Cliffs, NJ, 1986).
[17] L.C. Larson, *Problem Solving Through Problems* (Springer, New York, 1983).
[18] R.C. Linger, H.D. Mills and B.I. Witt, *Structured Programming: Theory and Practice* (Addison-Wesley, Reading, MA, 1979).
[19] Z. Manna, *Mathematical Theory of Computation* (McGraw-Hill, New York, 1974).
[20] A. Mili, A relational approach to the design of deterministic programs, *Acta Inform.* **20** (1983) 315–329.
[21] A. Mili, J. Desharnais and F. Mili, Programming heuristics for the design of deterministic programs, *Acta Inform.* **24** (1987) 239–276.
[22] F. Mili, J. Desharnais and A. Mili, *On Program Construction: A Heuristic Relational Approach*, Laval University, Quebec, Canada, August 1990.
[23] H.D. Mills, V.R. Basili, J.D. Gannon and R.G. Hamlet, *Principles of Computer Programming: A Mathematical Approach* (Allyn and Bacon, Boston, MA, 1986).
[24] C. Morgan, *Constructing Programs from Specifications*, (Prentice-Hall Intl., London, 1989).
[25] R. Morris and B. Wegbreit, Program verification by Subgoal induction, in: R.T. Yeh, ed., *Current Trends in Programming Methodology, Vol. 2* (Prentice-Hall, New York, 1977).
[26] J. Riguet, Relations binaires, fermetures et correspondances de Galois, *Bull. Soc. Math. France* **76** (1948) 114–155.
[27] A. Tarski, On the calculus of relations. *J. Symbolic Logic* **6**(3) (1941) 73–89.