Relational Mathematics for Relative Correctness

⋆

Jules Desharnais[1], Nafi Diallo[2], Wided Ghardallou[3], Marcelo F. Frias[4], Ali Jaoua[5], and Ali Mili[2]

[1] Université Laval, Quebec City, Canada
[2] New Jersey Institute of Technology, Newark, NJ, USA
[3] University of Tunis El Manar, Tunis, Tunisia
[4] Instituto Tecnológico de Buenos Aires (ITBA), and CONICET, Argentina
[5] Qatar University, Qatar
jules.desharnais@ift.ulaval.ca, ncd8@njit.edu, wided.ghardallou@gmail.com, mfrias@itba.edu.ar, jaoua@qu.edu.qa, ali.mili@njit.edu

Abstract. In earlier work, we had presented a definition of software fault as being any feature of a program that admits a substitution that would make the program more-correct. This definition requires, in turn, that we define the concept of relative correctness, i.e., what it means for a program to be more-correct than another with respect to a given specification. In this paper we broaden our earlier definition to encompass non-deterministic programs, or non-deterministic representations of programs; also, we study the mathematical properties of the new definition, most notably its relation to the refinement ordering, as well as its algebraic properties with respect to the refinement lattice.

Keywords

Absolute correctness, relative correctness, refinement ordering, refinement lattice, faults, fault removal.

1 Introduction

1.1 What is a Program Fault?

Our work stems from trying to define what is a software fault; usually we characterize a fault at some location in a program as a feature of the program that differs from what we believe it should be at that location. But this characterization presumes that we know with great precision and great certainty what the program ought to be doing at every location throughout its source code. Needless to say, such a presumption is unrealistic, since it is difficult in general to have a

⋆ Acknowledgement: This publication was made possible by a grant from the Qatar National Research Fund, NPRP04-1109-1-174. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the QNRF.

precise, complete, vetted specification of the overall software product, much less a specification of every small part thereof. Also, it is very common to find cases where the same faulty behavior of the program can be traced back to more than one possible feature, involving more than one location in the source code. In [8] we had defined a fault in a software product as any feature (be it a lexical token, a statement, a condition, a contiguous block, a set of non-contiguous statements, etc) that admits a substitute that would make the program strictly more-correct, in a sense to be defined. Such a definition, once we decide what it means to be more-correct, has the advantage that it does not depend on a detailed knowledge of the design of the software product, and that it characterizes faults without making any assumption about whether other parts of the program are, or are not, correct. It is worth noting that this definition of a fault, like any definition we could think of, is based on an implicit level of granularity of the program; this level of granularity corresponds to the degree of precision with which we want to isolate faults. At one extreme in the scale of granularity, we could consider lexical tokens; at the opposite extreme, we could consider the whole program as a monolith; most programmers think of faults at the granularity level of an assignment statement or equivalent syntactic units.

1.2 Deterministic and Non-deterministic Programs

In [2] we briefly discuss the properties of relative correctness, and its implications for software engineering processes, such as software testing, software repair, software faultiness analysis and in [3] we discuss the implication of relative correctness for software design. In all of our discussions in [2, 3, 8], we consider deterministic programs. In this paper we wish to lift the hypothesis of determinacy, and define relative correctness in the broader context of possibly non-deterministic programs. One may want to ask: why do we need to define relative correctness for non-deterministic programs if most programming languages of interest are deterministic? There are several reasons why we may want to do so:

- We may want to apply the concept of relative correctness, not only to finished software products, but also to partially defined intermediate designs (as appear in a stepwise refinement process).
- Non-determinacy is a convenient tool to model deterministic programs whose detailed behavior is difficult to capture, unknown, or irrelevant to a particular analysis.
- We may want to reason about the relative correctness of programs without having to compute their functions is all their minute details.

As an illustration, we consider the space S defined by the following declarations:

```
a:  array [1..N] of itemtype; x:  itemtype;
low, high: 0..N+1;  found:  boolean;
```

where `itemtype` is some data type that represents an ordered set, and we consider the following specification R and program `P`:

$R = \{(s, s') | found' = (\exists i : 1 \leq i \leq N : x = a[i])\}$,

```
P:  {low=2; high=N;  found=false;
     while (low<=high)
        {indextype m=(low+high)/2;
         if (x<a[m]) {high=m-1;}
         else  if (x>a[m]) {low=m+1;}
               else {found=true; low=m+1; high=m-1;}}}
```

We would like to think of the statement `low=2` as a fault, and that replacing this statement by `low=1` would produce a more-correct program; but to prove these claims using the original definition of relative correctness, we would have to compute the function of this program, i.e. determine the final values of all the program variables as a function of the initial values. But computing the final values of variables `low` and `high` is at the same time very difficult (as they depend on the position of `x` with respect to the array cells) and rather irrelevant (as they play an auxiliary role with respect to the function of the program). The interest of non-deterministic relations is that they enable us to focus on relevant functional aspects of a program, at the exclusion of complex and/or uninteresting details.

In section 2 we introduce some relational definitions and notations, which we use in section 3 to introduce a definition of relative correctness for non-deterministic programs; and in sections 4 and 5 we explore the properties of relative correctness, most notably its relation to the refinement ordering (section 4) and its relation to the refinement lattice (section 5). Finally, in section 6 we summarize our findings, compare them to related work, and sketch directions for future research.

2 Mathematics for Program Analysis

2.1 Relational Notations

In this section, we introduce some elements of relational mathematics that we use in the remainder of the paper to carry out our discussions. We assume the reader familiar with relational algebra, and we generally adhere to the definitions and notations of Brink et al. [1]. Dealing with programs, we represent sets using a programming-like notation, by introducing variable names and associated data type (sets of values). For example, if we represent set S by the variable declarations

$x : X; y : Y; z : Z,$

then S is the Cartesian product $X \times Y \times Z$. Elements of S are denoted in lower case s, and are triplets of elements of X, Y, and Z. Given an element s of S, we represent its X-component by $x(s)$, its Y-component by $y(s)$, and its Z-component by $z(s)$. When no risk of ambiguity exists, we may write x to represent $x(s)$, and x' to represent $x(s')$, letting the references to s and s' be implicit.

A relation on S is a subset of the Cartesian product $S \times S$; given a pair (s, s') in R, we say that s' is an *image* of s by R. Special relations on S include

the *universal* relation $L = S \times S$, the *identity* relation $I = \{(s, s')|s' = s\}$, and the *empty* relation $\phi = \{\}$. Operations on relations (say, R and R') include the set theoretic operations of *union* $(R \cup R')$, *intersection* $(R \cap R')$, *difference* $(R \setminus R')$ and *complement* $(\overline{R})$. They also include the *relational product*, denoted by $(R \circ R')$, or $(RR'$, for short) and defined by:

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

The *power* of relation R is denoted by R^n, for a natural number n, and defined by $R^0 = I$, and for $n > 0$, $R^n = R \circ R^{n-1}$. The *reflexive transitive closure* of relation R is denoted by R^* and defined by $R^* = \{(s, s')|\exists n \geq 0 : (s, s') \in R^n\}$. The *converse* of relation R is the relation denoted by $\widehat{R}$ and defined by

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

Finally, the *domain* of a relation R is defined as the set $dom(R) = \{s|\exists s' : (s, s') \in R\}$, and the *range* of relation R is defined as the domain of $\widehat{R}$.

A relation R is said to be *reflexive* if and only if $I \subseteq R$, *antisymmetric* if and only if $(R \cap \widehat{R}) \subseteq I$, *asymmetric* if and only if $(R \cap \widehat{R}) = \phi$, and *transitive* if and only if $RR \subseteq R$. A relation is said to be a *partial ordering* if and only if it is reflexive, antisymmetric, and transitive. Also, a relation R is said to be *total* if and only if $I \subseteq R\widehat{R}$, and a relation R is said to be *deterministic* (or: a *function*) if and only if $\widehat{R}R \subseteq I$. In this paper we use a property to the effect that two functions f and f' are identical if and only if $f \subseteq f'$ and $f'L \subseteq fL$. A relation R is said to be a *vector* if and only if $RL = R$; a vector on space S is a relation of the form $R = A \times S$, for some subset A of S; we use vectors to represent subsets of S, and we may by abuse of notation write $s \in R$ to mean $s \in A$; in particular, we use the product RL as a relational representation of the domain of R.

2.2 A Refinement Calculus

Throughout this paper, we interpret relations as program specifications or as programs and we may use the same symbol to refer to a program and to the relation that the program defines on its space. Given two relations R and R', we say that R' *refines* R (abbrev: $R' \sqsupseteq R$) if and only if: $RL \cap R'L \cap (R \cup R') = R$, where L is the universal relation and concatenation of relation names represents the product of the relations. We find that this condition is equivalent to: $RL \subseteq R'L \wedge RL \cap R' \subseteq R$. We also find that the refinement relation is reflexive, antisymmetric and transitive, and that it has lattice-like properties, in the following sense:

- Any two relations R and R' have a greatest lower bound, which we denote by $R \sqcap R'$ and to which we refer as the *meet* of R and R'. Also, we find: $R \sqcap R' = RL \cap R'L \cap (R \cup R')$.
- Given two relations R and R', we define the *join* of R and R' (denoted by $R \sqcup R'$) as:
$$(\overline{RL} \cap R') \cup (\overline{R'L} \cap R) \cup (R \cap R').$$

– Two relations R and R' admit a least upper bound if and only if they satisfy the condition $RL \cap R'L = (R \cap R')L$, which we call the *consistency condition*.
– Any two relations that satisfy the consistency condition admit a least upper bound, which is their join. In other words, we can always compute the join of two relations, but it equals their least upper bound only if they meet the consistency condition.

Let R and P be two relations on space S; we say that P (interpreted as a possibly non-deterministic program) is *correct* with respect to R (interpreted as a specification) if and only if P refines R. We have a proposition (due to [9]) to the effect that if P is deterministic, then P is correct with respect to R if and only if $(R \cap P)L = RL$.

3 Relative Correctness of Non-Deterministic Programs

3.1 Background

In this section, we briefly summarize our main findings with regards to deterministic programs, so as to convey our expectations with respect to non-deterministic programs. All our discussions about correctness, relative correctness, and faults, refer to a relational specification, which we usually denote by R. We denote candidate programs by P and P', and for the sake of convenience we make no distinction between a program (as a syntactic representation) and the function or relation that the program defines on its space. Given a program P and a specification R, we find that the domain of $R \cap P$ represents the set of initial states for which P delivers an output that is considered correct with respect to R; we refer to this set as the *competence domain* of P with respect to R. A (deterministic) program P' is said to be more-correct than a (deterministic) program P with respect to specification R if and only if it has a larger competence domain; we denote this by, $P' \sqsupseteq_R P$. Then we define a fault f in a program P as any feature of the program that admits a substitute that would make the program *strictly* more-correct (i.e. yield a strictly larger competence domain). Among the most salient properties we have found for the property of relative correctness, we cite:

– *Relative correctness is reflexive and transitive but not antisymmetric.* Reflexivity and transitivity stem from the reflexivity and transitivity of set inclusion, as it applies to competence domains. Relative correctness is not antisymmetric because programs may have the same competence domain and still be distinct, due to the non-determinacy of specifications. This property holds for non-deterministic programs (and the non-deterministic version of relative correctness), as we see in proposition 3.3.
– *Relative correctness culminates in absolute correctness.* A correct program with respect to specification R is more-correct with respect to R than any candidate program. Indeed, since we find (section 2.2) that a correct P satisfies the condition $dom(R \cap P) = dom(R)$, then a correct program has a

maximal competence domain. This property holds for non-deterministic programs (and the non-deterministic version of relative correctness), as we see in proposition 4.2.

- *Relative correctness logically implies enhanced reliability.* We find that if a program is more-correct than another, then it is necessarily more reliable. Indeed, if we measure reliability by the probability of successful execution modulo some probability distribution θ of input states, then the probability of successful execution of a program P modulo probability distribution θ is the integral (or for discrete models, the sum) of θ over the competence domain of P; clearly, the larger the competence domain the higher the probability. We do not prove that this property survives the transition to non-deterministic programs, though we suspect that it does, modulo an angelic interpretation of competence domain (whereby a non deterministic program is considered to behave correctly as soon as it provides at least one correct outcome with respect to R).
- *Relative Correctness and Refinement.* One of the most interesting properties we have found about relative correctness is its relationship to refinement. In [8], we find that a program P' refines a program P if and only if P' is more-correct than P with respect to any specification. Formally,

$$P' \sqsupseteq P \Leftrightarrow (\forall R : P' \sqsupseteq_R P).$$

  This property does not hold for non-deterministic programs (and the non-deterministic version of relative correctness), but we have an interesting substitute in proposition 4.3.

3.2 Definitions

The purpose of this section is to define the concept of relative correctness for arbitrary programs, that are not necessarily deterministic.

In seeking to generalize the property of relative correctness to non-deterministic programs, we consider two requirements: first, the formula for non-deterministic programs must be equivalent to the formula we already have for deterministic programs when the programs are deterministic; second, we wish to preserve the properties we have listed above, most notably the relation between relative correctness and refinement. We submit the following definition.

Definition 3.1. *We let R be a specification on set S and we let P and P' be (possibly non-deterministic) programs on space S. We say that P' is more-correct than P with respect to R (abbrev: $P' \sqsupseteq_R P$) if and only if:*

$$(R \cap P)L \subseteq (R \cap P')L \wedge (R \cap P)L \cap \overline{R} \cap P' \subseteq P.$$

Interpretation: P' is more-correct than P with respect to R if and only if it has a (n equal or) larger competence domain, and for the elements in the competence domain of P, program P' has (the same or) fewer images that violate R

than P does. Even though a more appropriate name for this relation is *at-least-as-correct-as*, we use the shorter version *more-correct-than*. As an illustration, we consider the set $S = \{0, 1, 2, 3, 4, 5, 6, 7\}$ and we let R, P and P' be defined as follows:

$R = \{(0,0), (0,1), (1,0), (1,1), (1,2), (2,1), (2,2), (2,3), (3,2), (3,3), (3,4)$
$(4,3), (4,4), (4,5), (5,4), (5,5), (5,6), (6,5), (6,6), (6,7), (7,6), (7,7)\}$

$P = \{(0,2), (0,3), (1,3), (1,4), (2,0), (2,1), (3,1), (3,2), (4,2), (4,3), (5,3),$
$(5,4), (6,3), (6,4), (7,4), (7,5)\}$

$P' = \{(0,2), (0,3), (1,2), (1,3), (2,0), (2,3), (3,1), (3,4), (4,2), (4,5), (5,3),$
$(5,6), (6,4), (6,7), (7,4), (7,5)\}$

From these definitions, we compute:

$R \cap P = \{(2,1), (3,2), (4,3), (5,4)\}$, $(R \cap P)L = \{2,3,4,5\} \times S$,
$R \cap P' = \{(1,2), (2,3), (3,4), (4,5), (5,6), (6,7)\}$
$(R \cap P')L = \{1,2,3,4,5,6\} \times S$
$(R \cap P)L \cap P' = \{(2,0), (2,3), (3,1), (3,4), (4,2), (4,5), (5,3), (5,6)\}$
$(R \cap P)L \cap \overline{R} \cap P' = \{(2,0), (3,1), (4,2), (5,3)\}$

We leave it to the reader to check that the two clauses of Definition 3.1 are satisfied. Figure 1 represents relations R, P and P' on space S. Program P' is more-correct than program P with respect to R because it has a larger competence domain ($\{1,2,3,4,5,6\}$ vs. $\{2,3,4,5\}$, highlighted in Figure 1) and because on the competence domain of P, program P' generates no incorrect output unless P also generates it.

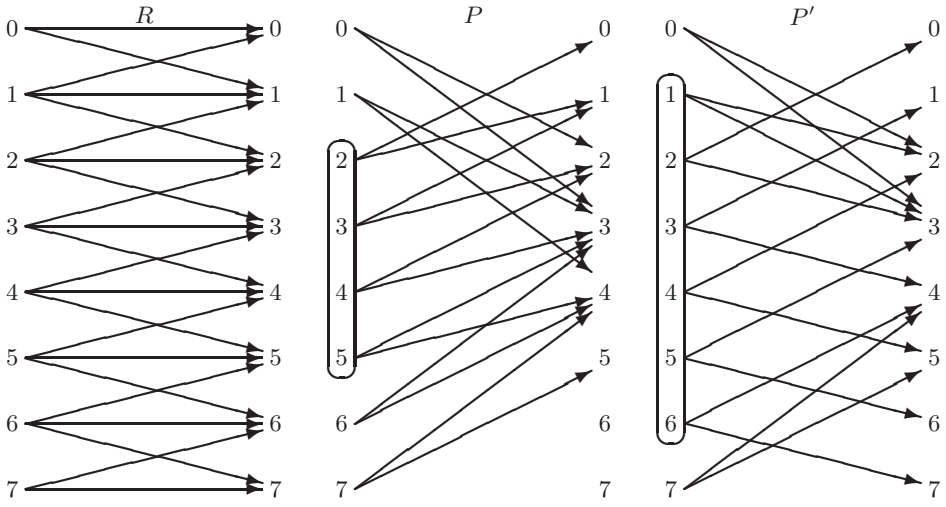

Fig. 1. Relative Correctness for Non-Deterministic Programs: $P' \sqsupseteq_R P$.

As a second illustration, we consider the binary search program introduced above, and we capture its semantics by the following relation:

$$P = \{(s, s') | found' = (\exists i : 2 \leq i \leq N : x = a[i])\}.$$

We let P' be the program obtained from P by replacing `low=2` by `low=1`, and we find:

$$P' = \{(s, s') | found' = (\exists i : 1 \leq i \leq N : x = a[i])\}.$$

We find easily that P' is more-correct than P with respect to R (where R is the search specification introduced in section 1.2): the first clause stems vacuously from $RL = P'L$ and $RL = L$, and the second clause stems vacuously from $R = P'$ (whence $\overline{R} \cap P' = \phi$).

3.3 Properties

The first property we want to check about this definition is that it generalizes the definition given in [8], which provides that a deterministic program P' is more-correct than a deterministic program P with respect to a specification R if and only if $(R \cap P)L \subseteq (R \cap P')L$. We have the following proposition.

Proposition 3.2. *Let R, P and P' be relations on S. If P' is deterministic then the conditions $(R \cap P)L \subseteq (R \cap P')L$ and $P' \sqsupseteq_R P$ are logically equivalent.*

Proof. The condition $P' \sqsupseteq_R P$ clearly implies $(R \cap P)L \subseteq (R \cap P')L$, hence we focus our attention on the reverse implication. We let P' be a function, we assume that P and P' satisfy the condition $(R \cap P)L \subseteq (R \cap P')L$, and we aim to prove the condition $(R \cap P)L \cap \overline{P} \cap P' \subseteq P$. We proceed as follows:

$(R \cap P)L \cap \overline{R} \cap P' \subseteq P$
$\Leftarrow$ {since $(R \cap P)L \subseteq (R \cap P')L$}
$(R \cap P')L \cap \overline{R} \cap P' \subseteq P$
$\Leftarrow$ {Boolean algebra}
$(R \cap P')L \cap P' \subseteq R$
$\Leftarrow$ {Dedekind}
$(R \cap P' \cap P'L)(L \cap (R \widehat{\cap} P')P') \subseteq R$
$\Leftarrow$ {Boolean algebra}
$(R \cap P')(R \widehat{\cap} P')P' \subseteq R$
$\Leftarrow$ {Boolean algebra, monotonicity}
$R \widehat{P'} P' \subseteq R$
$\Leftarrow$ {P' is deterministic, hence $\widehat{P'}P' \subseteq I$}
$R \subseteq R$,

which is a tautology. **qed**

As we see, this proof assumes that P' is deterministic but imposes no condition on P: indeed the second clause in the definition of relative correctness imposes a condition restricting the possible incorrect behavior of P' on the competence domain of P, where P' is known to behave correctly (since it has a

larger competence domain than P). Because P' is deterministic, it assigns only one image to any element of the competence domain of P, which is known to be a correct image; hence there is no scope for P' to associate an incorrect image. So that if P' satisfies the first clause of the definition of relative correctness and is deterministic, then it necessarily satisfies the second clause, regardless of relation P.

Proposition 3.3. *The relative correctness relation with respect to a given specification is reflexive and transitive.*

Proof. Reflexivity is trivial. To prove transitivity, we consider relations R, P, P' and P'', and we assume that P' is more-correct than P with respect to R, and that P'' is more-correct than P' with respect to R. The condition $(R \cap P)L \subseteq (R \cap P'')L$ stems readily from the hypothesis. We focus on the condition $(R \cap P)L \cap \overline{R} \cap P'' \subseteq P$, which we prove as follows:

$$
\begin{array}{ll}
 & (R \cap P)L \cap \overline{R} \cap P'' \subseteq P \\
\Leftrightarrow & \qquad \{\text{Hypothesis: } P' \sqsupseteq_R P\} \\
 & (R \cap P)L \cap (R \cap P')L \cap \overline{R} \cap P'' \subseteq P \\
\Leftrightarrow & \qquad \{\text{Hypothesis: } P'' \sqsupseteq_R P'\} \\
 & (R \cap P)L \cap (R \cap P')L \cap \overline{R} \cap P'' \cap P' \subseteq P \\
\Leftarrow & \qquad \{\text{Boolean algebra}\} \\
 & (R \cap P)L \cap \overline{R} \cap P' \subseteq P,
\end{array}
$$

which holds, by hypothesis. **qed**

Since $\sqsupseteq_R$ is reflexive and transitive, it is a preorder; we use this preorder to define an equivalence relation, as follows:

Definition 3.4. *Two relations P and P' are said to be* equally correct *with respect to specification R (abbrev: $P \equiv_R P'$) if and only if $P \sqsupseteq_R P'$ and $P' \sqsupseteq_R P$.*

For deterministic relations P and P', equal correctness simply means having the same competence domain; the following proposition characterizes equal correctness for arbitrary (not necessarily deterministic) relations.

Proposition 3.5. *Let R, P and P' be arbitrary relations on space S. Then*

$$(P \equiv_R P') \Leftrightarrow (R \cap P)L = (R \cap P')L \wedge (R \cap P)L \cap \overline{R} \cap P = (R \cap P')L \cap \overline{R} \cap P'.$$

Proof. From $P \equiv_R P'$ we infer readily that P and P' have the same competence domain with respect to R. Also, from $(R \cap P)L \cap \overline{R} \cap P' \subseteq P$ and $(R \cap P)L = (R \cap P')L$ we infer:

$$(R \cap P')L \cap \overline{R} \cap P' \subseteq (R \cap P)L \cap \overline{R} \cap P.$$

By interchanging P and P' and combining the two results, we find:

$$(R \cap P')L \cap \overline{R} \cap P' = (R \cap P)L \cap \overline{R} \cap P.$$

The converse implication is trivial, if we replace equality by inclusion, and note that an intersection is a subset of its terms. **qed**

Proposition 3.5 characterizes equivalence classes of relation $\equiv_R$ as being relations that share a common competence domain and a common set of incorrect outputs with respect to specification R. The following proposition singles out a representative element of each class, and shows that it is the least refined element of the class.

Proposition 3.6. *Let R and P be relations on space S. Then $\rho_R(P) = (R \cap P)L \cap (R \cup P)$ is in the same equivalence class of $\equiv_R$ as P. Furthermore, $\rho_R(P)$ is the least refined element of its equivalence class.*

Proof. We check that P and $\rho_R(P)$ are equally correct.

$P \equiv_R \rho_R(P)$

$\Leftrightarrow$ {Proposition 3.5}

$(R \cap P)L = (R \cap (R \cap P)L \cap (R \cup P))L \wedge$

$(R \cap P)L \cap \overline{R} \cap P = (R \cap (R \cap P)L \cap (R \cup P))L \cap \overline{R} \cap (R \cap P)L \cap (R \cup P)$

$\Leftrightarrow$ {Relational identities}

$(R \cap P)L = (R \cap P)L \wedge (R \cap P)L \cap \overline{R} \cap P = (R \cap P)L \cap \overline{R} \cap P$

which is a tautology. As for proving that $\rho_R(P)$ is the least refined element of its class, we let P' be an element in the equivalence class of P, and we show that P' refines $\rho_R(P)$.

$P' \equiv_R P$

$\Leftrightarrow$ {Proposition 3.5}

$(R \cap P)L = (R \cap P')L \wedge (R \cap P)L \cap \overline{R} \cap P = (R \cap P')L \cap \overline{R} \cap P'$

$\Rightarrow$ {Boolean algebra}

$(R \cap P)L = (R \cap P')L \wedge (R \cap P')L \cap \overline{R} \cap P' \subseteq P$

$\Leftrightarrow$ {Shunting}

$(R \cap P)L = (R \cap P')L \wedge (R \cap P')L \cap P' \subseteq R \cup P$

$\Rightarrow$ {P and P' have the same competence domain}

$(R \cap P)L \subseteq P'L \wedge (R \cap P)L \cap P' \subseteq (R \cap P)L \cap (R \cup P)$

$\Leftrightarrow$ {Substitution of $\rho_R(R)$, $\rho_R(P)L = (R \cap P)L$}

$\rho_R(P)L \subseteq P'L \wedge \rho_R(P)L \cap P' \subseteq \rho_R(P)$

$\Leftrightarrow$ {Definition of refinement}

$P' \sqsupseteq \rho_R(P)$. **qed**

It stems from this proposition that if P and P' are equally correct with respect to some specification R, then $\rho_R(P)$ and $\rho_R(P')$ are identical. Figure 2 shows an example of a specification P, two equally correct programs with respect to R, P and P', and the least refined relation of their shared equivalence class. The reader may check that P and P' are both refinements of $\rho_R(P)$ ($=\rho_R(P')$).

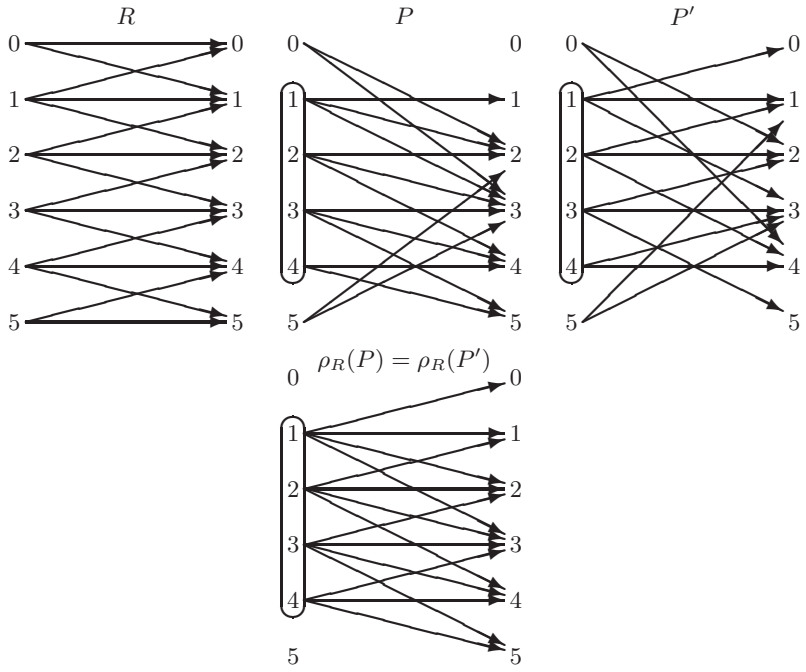

Fig. 2. Equal Correctness of Non-Deterministic Programs: $P' \equiv_R P$.

4 Relative Correctness and Refinement

Because refinement plays a central role in the definition of (absolute) correctness, it is legitimate to explore the relationship between refinement and *relative* correctness; this is the subject of this section.

Proposition 4.1. *Let R, P and P' be relations on set S. Then P' is more-correct than P with respect to R if and only if $\rho_R(P')$ refines $\rho_R(P)$, i.e.*

$$P' \sqsupseteq_R P \Leftrightarrow \rho_R(P') \sqsupseteq \rho_R(P).$$

Proof. We proceed by equivalences:

$\rho_R(P') \sqsupseteq \rho_R(P)$
$\Leftrightarrow$ {Formula of refinement}
$\rho_R(P)L \subseteq \rho_R(P')L \wedge \rho_R(P)L \cap \rho_R(P') \subseteq \rho_R(P)$
$\Leftrightarrow$ {Substitutions, and $\rho_R(P)L = (R \cap P)L$}
$(R \cap P)L \subseteq (R \cap P')L \wedge (R \cap P)L \cap (R \cap P')L \cap (R \cup P') \subseteq (R \cap P)L \cap (R \cup P)$
$\Leftrightarrow$ {$A \subseteq B \Leftrightarrow (A \cap B = A)$}
$(R \cap P)L \subseteq (R \cap P')L \wedge (R \cap P)L \cap (R \cup P') \subseteq (R \cap P)L \cap (R \cup P)$
$\Leftrightarrow$ {$(A \cap B \subseteq A \cap C) \Leftrightarrow (A \cap B \subseteq C)$}
$(R \cap P)L \subseteq (R \cap P')L \wedge (R \cap P)L \cap (R \cup P') \subseteq R \cup P$
$\Leftrightarrow$ {Shunting, and Boolean algebra}

$$
\begin{array}{ll}
& (R \cap P)L \subseteq (R \cap P')L \wedge (R \cap P)L \cap P' \cap \overline{R} \subseteq P \\
\Leftrightarrow & \qquad \{\text{Substitution}\} \\
& P' \sqsupseteq_R P. \qquad \textbf{qed}
\end{array}
$$

The following proposition casts absolute correctness as the culmination of relative correctness, in the sense that a correct program is more-correct than (or as correct as) any candidate program.

Proposition 4.2. *Let R be a specification on set S, and let P' be a relation that represents a program on S. P' is correct with respect to R if and only if P' is more-correct with respect to R than any program P on S, i.e.*

$$P' \sqsupseteq R \Leftrightarrow (\forall P : P' \sqsupseteq_R P).$$

Proof. Proof of sufficiency: We assume that P' refines R and we let P be an arbitrary relation on S. We must show that P' is more-correct than P with respect to R, i.e. that

$$(R \cap P)L \subseteq (R \cap P')L \wedge (R \cap P)L \cap \overline{R} \cap P' \subseteq P.$$

We write:

$$
\begin{array}{ll}
& (R \cap P')L \\
\supseteq & \qquad \{\text{By hypothesis, } P' \cap RL \subseteq R\} \\
& (P' \cap RL \cap P')L \\
= & \qquad \{\text{Relational identity}\} \\
& RL \cap P'L \\
\supseteq & \qquad \{\text{Boolean algebra}\} \\
& (R \cap P')L.
\end{array}
$$

On the other hand,

$$
\begin{array}{ll}
& (R \cap P)L \cap \overline{R} \cap P' \\
\subseteq & \qquad \{\text{Monotonicity}\} \\
& RL \cap P' \cap \overline{R} \\
\subseteq & \qquad \{\text{By hypothesis, } P' \cap RL \subseteq R\} \\
& R \cap \overline{R} \\
\subseteq & \qquad \{\text{Since } R \cap \overline{R} = \phi\} \\
& P.
\end{array}
$$

Proof of Necessity: We assume that P' is more-correct than any relation P on S with respect to specification R, and we write this property for $P = R$. This yields:

$$
\begin{array}{ll}
& (R \cap R)L \subseteq (R \cap P')L \wedge (R \cap R)L \cap \overline{R} \cap P' \subseteq R \\
\Leftrightarrow & \qquad \{\text{Simplification and Shunting}\} \\
& RL \subseteq (R \cap P')L \wedge RL \cap P' \subseteq R \cup R \\
\Rightarrow & \qquad \{\text{Monotonicity}\} \\
& RL \subseteq P'L \wedge RL \cap P' \subseteq R \\
\Leftrightarrow & \qquad \{\text{Definition}\}
\end{array}
$$

$P' \sqsupseteq P$. **qed**

In [8], we find that for deterministic relations P and P', P' refines P if and only if P' is more-correct than P with respect to any specification. This property can be interpreted as follows: if P' refines P, then whatever P does, P' can do as well or better; in particular, P' is more-correct than P with respect to any specification R. In other words, the only way for P' to be more-correct than P with respect to any specification R is for P' to merely refine P. When P and P' are not necessarily deterministic, we find that the condition $(\forall R : P' \sqsupseteq_R P)$ is too strong a sufficient condition for $P' \sqsupseteq P$, and too strong a necessary condition. We have the following proposition.

Proposition 4.3. *Let P and P' be relations on set S. P' refines P if and only if P' is more-correct than P with respect to specification P.*

Proof. *Proof of Sufficiency.* From $P' \sqsupseteq_P P$, we infer $(P \cap P)L \subseteq (P \cap P')L$ and $(P \cap P)L \cap \overline{P} \cap P' \subseteq P$, which implies $PL \subseteq P'L$ and $PL \cap P' \subseteq P$.

Proof of Necessity. The condition $(P' \sqsupseteq_P P)$ can be written as:

$$PL \subseteq (P \cap P')L \wedge PL \cap \overline{P} \cap P' \subseteq P.$$

The second clause stems readily from the hypothesis, since $P' \sqsupseteq P$ implies $PL \cap P' \subseteq P$. As for the first clause, we prove it as follows:

$PL \cap P' \subseteq P$
$\Rightarrow$ {Taking the intersection with P'}
$PL \cap P' \subseteq P \cap P'$
$\Rightarrow$ {Multiplying by L}
$(PL \cap P')L \subseteq (P \cap P')L$
$\Rightarrow$ {Relational identity}
$PL \cap P'L \subseteq (P \cap P')L$
$\Rightarrow$ {Since $PL \subseteq P'L$}
$PL \subseteq (P \cap P')L$ **qed**

In other words, according to this proposition, P' does not have to be more-correct than P with respect to any specification; it suffices that it be more-correct than P with respect to a single specification, namely P itself. The interpretation of this proposition is quite straightforward: The property of P' to be more-correct than P with respect to P can be interpreted to mean that P' beats P at its own game; this sounds like a good characterization of refinement. The following example disproves that $P' \sqsupseteq P$ logically implies $(\forall R : P' \sqsupseteq_R P)$. We take:

$$S = \{0, 1\}, \quad R = \{(0, 1)\}, \quad P = \{(0, 0), (0, 1)\}, \quad P' = \{(0, 0)\}.$$

Indeed, P' clearly refines P. Yet P' is not more-correct than P with respect to R, as we can check by observing that: $(R \cap P) = \{(0, 1)\}$, hence $(R \cap P)L = \{(0, 0), (0, 1)\}$ and $(R \cap P') = \phi$, hence $(R \cap P')L = \phi$. While $(P' \sqsupseteq P)$ does not

logically imply that P' is more-correct than P for any relation R, it does imply than P' is more-correct than P with respect to a single relation, namely P.

Hence while in [8] we have found that for deterministic relations P and P', $P' \sqsupseteq P$ is equivalent to $(\forall R : P' \sqsupseteq_R P)$, Propositions 4.3 provides that for relations P and P' that are not necessarily deterministic, $P' \sqsupseteq P$ is equivalent to $(P' \sqsupseteq_P P)$. This means in particular that for deterministic P and P', $(P' \sqsupseteq_P P)$ logically implies $(\forall R : P' \sqsupseteq_R P)$. This is an intriguing property, but one that we can understand intuitively: if we take two arbitrary programs P and P', then P' could conceivably be more-correct than P with respect to some specification, and less-correct with respect to other specifications; but if P' is more-correct than P with respect to P *itself*, then P' clearly dominates P, i.e. there is nothing P could do that P' could not; this conveys the same idea of subsumption that we associate with refinement.

To conclude this section, we consider the following question: Is it possible that if P' is more-correct than P with respect to some relation R, then it is more-correct than P with respect to any relation that R refines? Intuitively, it sounds like it should since refinement reflects the strength of a specification; the following example shows that this is not the case. We consider:

$$S = \{0, 1, 2\}, \quad P = \{(0, 0)\}, \quad P' = \{(0, 2)\}, \quad R = \{(0, 1)\}, \quad Q = \{(0, 0), (0, 1)\}.$$

We do have $R \sqsupseteq Q$, and we do have $P' \sqsupseteq_R P$ since $(R \cap P)L = \phi \subseteq (R \cap P')L$ and $(R \cap P)L \cap \overline{R} \cap P' = \phi \subseteq P$. Yet, P' is not more-correct than P with respect to Q, since $(Q \cap P)L = \{(0, 0), (0, 1), (0, 2)\}$ whereas $(Q \cap P')L = \phi$.

5 Relative Correctness and Refinement Lattice

In section 2.2 we have introduced some lattice-like operators including the least upper bound and the greatest lower bound of two specifications; we have found that when two specifications R and Q satisfy the consistency condition $(RL \cap QL = (R \cap Q)L)$ then they admit a least upper bound. In this section we raise the question whether $(P' \sqsupseteq_R P)$ and $(P' \sqsupseteq_Q P)$ logically imply $(P' \sqsupseteq_{R \sqcup Q} P)$, where $(R \sqcup Q)$ is the least upper bound (modulo the refinement ordering) of R and Q. The following proposition gives a nuanced answer to this question.

Proposition 5.1. *Let P and P' be two programs (relations) on space S and let R and Q be two specifications on S. If P' is deterministic, and if it is more-correct than P with respect to R and with respect to Q then it is more-correct than P with respect to $(R \sqcup Q)$, i.e. $\widehat{P'}P' \subseteq I \wedge P' \sqsupseteq_R P \wedge P' \sqsupseteq_Q P \Rightarrow P' \sqsupseteq_{R \sqcup Q} P$.*

Proof. We introduce a lemma that will be useful for our proof:

$$\widehat{P}P \subseteq I \wedge Q \subseteq P \Rightarrow (R \cap P)L \cap Q = R \cap Q.$$

To this effect, we write:

$$\begin{aligned} & (R \cap P)L \cap Q = R \cap Q \\ \Leftrightarrow \quad & \{(R \cap P)L \cap Q \subseteq Q,\ R \cap Q \subseteq (R \cap P)L,\ R \cap Q \subseteq Q,\ Q \subseteq P\} \end{aligned}$$

$(R \cap P)L \cap Q \subseteq R$

$\Leftarrow$ {Dedekind}

$(R \cap P \cap QL)(L \cap (\widehat{R \cap P})Q) \subseteq R$

$\Leftarrow$ {Hypothesis: $Q \subseteq P$, and monotonicity of the product}

$(R \cap P)(\widehat{R \cap P})P \subseteq R$

$\Leftarrow$ {Monotonicity of converse and product}

$R\widehat{P}P \subseteq R$

$\Leftarrow$ {Monotonicity of product}

$\widehat{P}P \subseteq I$

$\Leftarrow$ {Hypothesis: $\widehat{P}P \subseteq I$}

true.

Using this lemma, we now show that if P' is more-correct than P with respect to R and with respect to Q, then it is more-correct than P with respect to $(R \sqcup Q)$. We write:

$P' \sqsupseteq_{Q \sqcup R} P$

$\Leftrightarrow$ {Proposition 3.2}

$((Q \sqcup R) \cap P)L \subseteq ((Q \sqcup R) \cap P')L$

$\Leftrightarrow$ {Definition of $\sqcup$}

$(((\overline{RL} \cap Q) \cup (\overline{QL} \cap R) \cup (R \cap R)) \cap P)L$
$\subseteq (((\overline{RL} \cap Q) \cup (\overline{QL} \cap R) \cup (R \cap R)) \cap P')L$

$\Leftrightarrow$ {Distributing L, Boolean algebra, and $(PL \cap Q)R = PL \cap QR$}

$(\overline{RL} \cap (Q \cap P)L) \cup (\overline{QL} \cap (R \cap P)L) \cup (Q \cap R \cap P)L$
$\subseteq (\overline{RL} \cap (Q \cap P')L) \cup (\overline{QL} \cap (R \cap P')L) \cup (Q \cap R \cap P')L$

$\Leftarrow$ {Boolean algebra, hypotheses $P' \sqsupseteq_Q P$ and $P' \sqsupseteq_R P$}

$(Q \cap R \cap P)L \subseteq (Q \cap R \cap P')L$

$\Leftarrow$ {For any relations A and B, $(A \cap B)L \subseteq AL \cap BL$}

$(Q \cap P)L \cap (R \cap P)L \subseteq (Q \cap R \cap P')L$

$\Leftarrow$ {By hypothesis, $(Q \cap P)L \subseteq (Q \cap P')L$ and $(R \cap P)L \subseteq (R \cap P')L$}

$(Q \cap P')L \cap (R \cap P')L \subseteq (Q \cap R \cap P')L$

$\Leftrightarrow$ {Rewriting the first L as LL and factoring L}

$((Q \cap P')L \cap (R \cap P'))L \subseteq (Q \cap R \cap P')L$

$\Leftarrow$ {lemma, using P', $(R \cap P')$, Q for P, Q, R}

$(Q \cap R \cap P')L \subseteq (Q \cap R \cap P')L$

$\Leftrightarrow$ {Tautology}

true. **qed**

This result holds regardless of whether R and Q satisfy the consistency condition: if R and Q do, then this result pertains for their least upper bound; if not, then the result pertains for their join, which is not their least upper bound. To prove that the condition of determinacy of P' is a necessary condition in proposition 5.1, we consider the following (counter) example on set $S = \{0, 1, 2\}$ where P' is not deterministic, and we prove that then P' can be more-correct than P with respect to two specifications without being more-correct with respect to

their join:

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

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

Indeed, we find $(P \cap R)L = (P' \cap R)L = \{(0,0),(0,1),(0,2)\}$ and $(R \cap P)L \cap \overline{R} \cap P' = \{(0,1)\}$, which is a subset of P, hence $P' \sqsupseteq_R P$. On the other hand, we find $(P \cap Q)L = (P' \cap Q)L = \{(0,0),(0,1),(0,2)\}$ and $(P \cap Q)L \cap \overline{Q} \cap P' = \{(0,2)\}$, which is a subset of P, hence $P' \sqsupseteq_Q P$. And yet, $(R \sqcup Q) = \{(0,0)\}$, whence $(P' \cap (R \sqcup Q)) = \phi$; therefore P' is not more-correct than P with respect to $(R \sqcup Q)$.

Whereas proposition 5.1 elucidates how relative correctness distributes over the join, the following proposition explores the same property for the meet.

Proposition 5.2. *If P' is more-correct than P with respect to R and with respect to Q, then it is more-correct than P with respect to $(R \sqcap Q)$.*

Proof. $P' \sqsupseteq_{R \sqcap Q} P$

$\Leftrightarrow$ {Definition of relative correctness}

$((Q \sqcap R) \cap P)L \subseteq ((Q \sqcap R) \cap P')L \wedge ((Q \sqcap R) \cap P')L \cap \overline{(Q \sqcap R)} \cap P' \subseteq P$

$\Leftrightarrow$ {Definition of meet}

$(QL \cap RL \cap (Q \cup R) \cap P)L \subseteq (QL \cap RL \cap (Q \cup R) \cap P')L$

$\wedge (QL \cap RL \cap (Q \cup R) \cap P)L \cap \overline{QL \cap RL \cap (Q \cup R)} \cap P' \subseteq P$

$\Leftrightarrow$ {Distribution, De Morgan}

$(RL \cap Q \cap P)L \cup (QL \cap R \cap P)L \subseteq (RL \cap Q \cap P')L \cup (QL \cap R \cap P')L$

$\wedge (QL \cap RL \cap (Q \cup R) \cap P)L \cap (\overline{QL} \cup \overline{RL} \cup (\overline{Q} \cap \overline{R})) \cap P' \subseteq P$

$\Leftarrow$ {Distribution, Boolean algebra}

$(Q \cap P)L \subseteq (Q \cap P')L \wedge (R \cap P)L \subseteq (R \cap P')L$

$\wedge ((Q \cap P)L \cap \overline{Q} \cap P') \cup ((R \cap P)L \cap \overline{R} \cap P') \subseteq P$

$\Leftarrow$ {Definition of relative correctness}

$P' \sqsupseteq_Q P \wedge P' \sqsupseteq_R P.$ **qed**

6 Concluding Remarks

6.1 Summary

In [8] we have introduced the concept of relative correctness as it applies to deterministic programs, and have used it to define the concept of a fault in a program with respect to a specification. In this paper, we generalize the definition of relative correctness to non-deterministic programs, on the grounds that very often, even when we are dealing with deterministic programs, we may want to reason about relative correctness without having to compute the functions of candidate programs in all their minute detail. To this effect, we introduce a definition, investigate its properties, and explore its relation to refinement as well as its algebraic properties with respect to lattice operations.

6.2 Prospects

One of the broadest venues of research that this paper opens pertains to the approximation of determinstic programs by non-deterministic relations. If we approximate program P by relation Π and program P' by relation Π', what relation must hold between P and Π, and between P' and Π', in order for a conclusion we draw on Π and Π' to carry over to P and P'. Interestingly, such a relation must necessarily involve R, the specification against which we define relative correctness. As an example, let P and P' be two programs on some space S defined by two variables, say x and y, and let R be the following specification on S:

$$R = \{(s, s') | y' = f(y)\},$$

for some function f. Clearly, we can reason about the relative correctness of P and P' by considering abstractions thereof, say Π and Π', that focus exclusively on variable y. We want to generalize this argument by characterizing the relation that must hold between P, P', Π, Π' and R so that we can analyze Π and Π' and infer conclusions about the relative correctness of P and P' with respect to R. This is currently under investigation.

6.3 Related Work

Several authors have introduced and studied concepts that are similar to relative correctness, and some refer to them by this exact name [4–7, 10, 11]. In [7] Logozzo discusses a framework for ensuring that some semantic properties are preserved by program transformation in the context of software maintenance. In [4] Lahiri et al. present a technique for verifying the relative correctness of a program with respect to a previous version, where they represent specifications by means of executable assertions placed throughout the program, and they define relative correctness by means of inclusion relations between sets of successful traces and unsuccessful traces. Logozzo and Ball [6] take a similar approach to Lahiri et al. in the sense that they represent specifications by a network of executable assertions placed throughout the program, and they define relative correctness in terms of successful traces and unsuccessful traces of candidate programs; Logozzo and Ball distinguish between two categories of program failures, namely contract violations when functional requirements are violated and run-time errors, when operational requirements are violated. In [10], Nguyen et al. present an automated repair method based on symbolic execution, constraint solving, and program synthesis; they call their method SemFix, on the grounds that it performs program repair by means of semantic analysis. In [11], Weimer et al. discuss an automated program repair method that takes as input a faulty program, along with a set of positive tests (i.e. test data on which the program is known to perform correctly) and a set of negative tests (i.e. test data on which the program is known to fail) and returns a set of possible patches. In [5] Le Goues et al. survey existing technology in automated program repair and identify open research challenges; among the criteria for automated repair methods, they cite applicability (extent of real-world relevance), scalability (ability to operate

effectively and efficiently for products of realistic size), generality (scope of application domain, types of faults repaired), and credibility (extent of confidence in the soundness of the repair tool).

Our work differs significantly from all these works in many ways:

- First, we use relational specifications that address the functional properties of the program as a whole, and have no cognizance of intermediate assertions that are expected to hold throughout the program; also, our relational specifications do not necessarily correspond to an abstraction of the assertions used in trace-based program analysis, because the initial and final assertion could be checking some local properties, whereas our specifications capture global input/ output properties.
- Second, our definition of relative correctness involves competence domains (for deterministic specifications) and the sets of states that candidate programs produce in violation of the specification (for non-deterministic programs).
- Third we conduct a detailed analysis of the relations between relative correctness and the property of refinement.
- Finally, we study how the property of relative correctness can be decomposed using lattice operators on the reference specification.

Acknowledgements

The authors are very grateful to the anonymous reviewers for their valuable feedback, which has greatly improved the form and content of this paper.

References

[1] Ch. Brink, W. Kahl, and G. Schmidt. *Relational Methods in Computer Science*. Springer Verlag, January 1997.

[2] Nafi Diallo, Wided Ghardallou, and Ali Mili. Correctness and relative correctness. In *Proceedings, 37th International Conference on Software Engineering*, Firenze, Italy, May 20–22 2015.

[3] Nafi Diallo, Wided Ghardallou, and Ali Mili. Program derivation by correctness enhancements. In *Refinement 2015*, Oslo, Norway, June 2015.

[4] Shuvendu K. Lahiri, Kenneth L. McMillan, Rahul Sharma, and Chris Hawblitzel. Differential assertion checking. In *Proceedings, ESEC/ SIGSOFT FSE*, pages 345–455, 2013.

[5] Claire LeGoues, Stephanie Forrest, and Westley Weimer. Current challenges in automatic software repair. *Software Quality Journal*, 21(3):421–443, 2013.

[6] Francesco Logozzo and Thomas Ball. Modular and verified automatic program repair. In *Proceedings, OOPSLA*, pages 133–146, 2012.

[7] Francesco Logozzo, Shuvendu Lahiri, Manual Faehndrich, and San Blackshear. Verification modulo versions: Towards usable verification. In *Proceedings, PLDI*, 2014.

[8] Ali Mili, Marcelo Frias, and Ali Jaoua. On faults and faulty programs. In Peter Hoefner, Peter Jipsen, Wolfram Kahl, and Martin Eric Mueller, editors, *Proceedings, RAMICS: 14th International Conference on Relational and Algebraic Methods in Computer Science*, volume 8428 of *Lecture Notes in Computer Science*, Marienstatt, Germany, April 28–May 1st 2014. Springer.

[9] H.D. Mills, V.R. Basili, J.D. Gannon, and D.R. Hamlet. *Structured Programming: A Mathematical Approach*. Allyn and Bacon, Boston, Ma, 1986.

[10] Hoang Duong Thien Nguyen, DaWei Qi, Abhik Roychoudhury, and Satish Chandra. Semfix: Program repair via semantic analysis. In *Proceedings, ICSE*, pages 772–781, 2013.

[11] Weimer W., Nguyen T., Le Goues C., and Forrest S. Automatically finding patches using genetic programming. In *Proceedings, ICSE 2009*, pages 364–374, 2009.