PUC

Monografias em Ciência da Computação
nº 12/90

Partial Relations for Program Derivation - Adequacy, Inevitability and Expressiveness

Armando M. Haeberer
Paulo A. S. Veloso

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Monografias em Ciência da Computação, Nº 12/90
Editor: Carlos J. P. Lucena Julho, 1990

Partial Relations for Program Derivation - Adequacy, Inevitability and Expressiveness *

Armando M. Haeberer
Paulo A. S. Veloso

* This work has been sponsored by the Secretaria de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

Partial Relations for Program Derivation
Adequacy, Inevitability and Expressiveness

A. M. Haeberer and P. A. S.Veloso

Depto. Informática, Pont. Univ. Católica do Rio de Janeiro
Rua Mq. de S. Vicente 225; Rio de Janeiro, RJ 22453; Brasil

Abstract

A framework for specifying, and reasoning about, problems and programs is presented. This framework is based on partial binary relations (relations together with two carriers sets) and underlies the semantics of any formalism for reasoning about programs together with preconditions. An algebraic calculus is developed and argued to be adequate for program derivation by means of formal manipulation of expressions. Its expressive power is shown to encompass that of first-order logic, thereby solving Tarski's problem on the expressiveness of his calculus of binary relations. Although our problem-theoretic formalism is based on partial binary relations, our considerations are perfectly general.

1. INTRODUCTION

This paper presents a framework, based on *partial binary relations* (relations together with two carriers sets), for specifying, and reasoning about, problems and programs. This framework underlies the semantics of any formalism for reasoning about programs together with preconditions. Based on our formalism we develop a calculus and we argue that its algebraic flavor is adequate for program derivation by means of formal manipulation of expressions. Furthermore, we prove that its expressive power encompasses that of first-order

ON THE REPRESENTABILITY OF THE ∇-ABSTRACT RELATIONAL ALGEBRA

G. BAUM - A. M. HAEBERER - P. A. S. VELOSO

The *∇-Abstract Relational Algebra* [Hae91, Vel91] is an abstract relational algebra $A = \langle \mathfrak{R}, +, 0, \bullet, \infty, ;, 1, \breve{\ } \rangle$ [Jón52] (where 0 and ∞ as the extremes of the Boolean reduct and 1 denoting the identity relation) extended by the operation ∇ which can be defined informally in terms of its correspondent proper relational algebra as: *given relations r and s, if xry (meaning the par* $\langle x, y\rangle \in r$*) and xsy then* $xr\nabla s[x,y]$. Where the pair $[x,y]$ belongs to the free grupoid $\langle V, [\ ]\rangle$, since ∇ is not associative.

The goal of such extension is to achieve the expressing power of firs-order calssical logic in order to obtain an appropriate tool for formal program construction.

As we know [Jón52] an abstract relational algebra is not *representable*, i.e., not all its models are isomorphous with a proper relational algebra (i.e. considering relations as being subsets of $V \times V$. In the sequel we will denote by $\wp$ the relation $\overline{1}$.

Defining $\Pi_1 = \overbrace{1\nabla\infty}$ and $\Pi_2 = \overbrace{\infty\nabla 1}$ (notice that $\overbrace{1\nabla\infty}$ denotes de *converse* of $1\nabla\infty$), the following properties of ∇ can be taken as some of the axioms defining it in the ∇-Abstract Relational Algebra, 1) $t \subset r\nabla s \rightarrow (t;\Pi_1 \subset r \wedge t;\Pi_2 \subset s)$, 2) $(r\nabla s);\left(\overbrace{t\nabla q}\right) = (r;\breve{t})\bullet(s;\breve{q})$ and then $(r\nabla s);\left(\overbrace{1\nabla 1}\right) = r\bullet s$, 3) $r;(s\nabla t) \subset (r;s)\nabla(r;t)$ and 4) $(r\subset v)\wedge(s\subset w) \rightarrow (r\nabla s)\subset(v\nabla w)$.

It is well known [Jón52] that an abstract relational is representable iff all its atoms are functional. Let us prove that *if $t \neq 0 \wedge t \subset r\nabla s \rightarrow (\exists v)(\exists w)(v\neq 0 \wedge w \neq 0 \wedge v\nabla w \subset t)$ holds, the ∇–Abstract Relational Algebra is representable by proving that the former condition implies the functionality of the atoms.*

Proposition a. If α is an atom then $\alpha\nabla\alpha$ is an atom.

Proof Take $0 \neq \beta \subset \alpha\nabla\alpha$. Since $t \neq 0 \wedge t \subset r\nabla s \rightarrow (\exists v)(\exists w)(v\neq 0 \wedge w\neq 0 \wedge v\nabla w \subset t)$ holds, there exists $\beta' \neq 0$ and $\beta'' \neq 0$ such that $\beta'\nabla\beta'' \subset \beta$.

Then, $\beta'\nabla\beta'' \subset \beta \subset \alpha\nabla\alpha$ (i)

By ; monotonicity we have,

$(\beta'\nabla\beta'');\Pi_1 \subset \beta;\Pi_1 \subset (\alpha\nabla\alpha);\Pi_1$ and $(\beta'\nabla\beta'');\Pi_2 \subset \beta;\Pi_2 \subset (\alpha\nabla\alpha);\Pi_2$

but, by (1) is $(\alpha\nabla\alpha);\Pi_1 \subset \alpha$ and $(\alpha\nabla\alpha);\Pi_2 \subset \alpha$ and since α is an atom and $(\alpha\nabla\alpha);\Pi_1 \neq 0$ and $(\alpha\nabla\alpha);\Pi_2 \neq 0$, both equals α.

Then, $(\beta'\nabla\beta'');\Pi_1 = \alpha$ and $(\beta'\nabla\beta'');\Pi_2 = \alpha$. Hence,

$(\beta'\nabla\beta'');\Pi_1 = (\beta'\nabla\beta'');\overbrace{1\nabla\infty} = \beta'\bullet(\beta'';\infty) = \alpha$ and

$(\beta'\nabla\beta'');\Pi_2 = (\beta'\nabla\beta'');\overbrace{\infty\nabla 1} = (\beta';\infty)\bullet\beta'' = \alpha$

By intersecting both sides of the above equalities we have,

$\beta'\bullet(\beta'';\infty)\bullet(\beta';\infty)\bullet\beta'' = \alpha$. So, $\beta'\bullet\beta'' = \alpha$ (ii)

But $\beta'\bullet\beta'' \subset \beta'$ and $\beta'\bullet\beta'' \subset \beta''$, then, by ∇ monotonicity (4) and recalling (i) is $(\beta'\bullet\beta'')\nabla(\beta'\bullet\beta'') \subset \beta'\nabla\beta'' \subset \beta \subset \alpha\nabla\alpha$, then, by (ii) we have,

$\alpha\nabla\alpha \subset \beta'\nabla\beta'' \subset \beta \subset \alpha\nabla\alpha$, hence, $\beta = \alpha\nabla\alpha$

Then, $\alpha\nabla\alpha$ is an atom since any relation included in it is either 0 or $\alpha\nabla\alpha$.

Proposition b. If r is an atom then $(r;\wp)\bullet r = 0$

Proof. From (2) we can deduce that $(r;\wp)\bullet r = (r\nabla r);\left(\breve{\wp\nabla 1}\right)$
By subdistributivity of ; with respect to ∇ (3), we have $r;(1\nabla 1)\subset r\nabla r$. But since by Proposition a $r\nabla r$ is an atom, we have $r;(1\nabla 1) = r\nabla r$ (since $r;(1\nabla 1)$ cannot be 0). Then $(r;\wp)\bullet r = r;(1\nabla 1);\left(\breve{\wp\nabla 1}\right) = r;\left((\wp\nabla 1);\left(\breve{1\nabla 1}\right)\right)$, which by (2) equals $r;(\wp\bullet 1)$, then $(r;\wp)\bullet r = 0$.

Proposition c. If r is an atom then $(\breve{r};r)\bullet\wp = 0$

Proof We know that $(r;s)\bullet t = (r;((\breve{r};t)\bullet s))\bullet t$ (exercise 2.3.11 [Sch89] then we have $(\breve{r};r)\bullet\wp = (\breve{r};((r;\wp)\bullet r))\bullet\wp$. But by Proposition b we know that if r is an atom then $(r;\wp)\bullet r = 0$ thus $(\breve{r};r)\bullet\wp = 0$.

But $(\breve{r};r)\bullet\wp = 0$ means that $(\breve{r};r)\subset 1$ which is equivalent to state that r is functional [Jón52]. Then we have proved that whenever r is an atom r is functional.

Thus, the ∇–Abstract Relational Algebra is representable.

References

[Jón52] Jónsson, B., Tarski, A., "Boolean Algebra with Operators", Amer. J. of Math. 74 (1952), pp. 85-97.

[Hae91] Haeberer, A. M., Veloso, P. A. S., "Partial Relations for Program Derivation: Adequacy, Inevitability and Expressiveness". In Möller, B. (ed) Constructing Programs from Specifications. North Holland, 1991.

[Sch89] Schmidt, G., Ströhlein, Th., "Relationen und Graphen", Springer-Verlag (1989), Reihe: Matematik für Informatiker 1.

[Vel91] Veloso, P. A. S., Haeberer, A. M., "A Finitary Relational Algebra for Classical First-Order Logic", Bulletin of the Section of Logic of the Polish Academy of Sciences, 20.2 (1991), pp. 52-62.

Möller, B., (ed) Constructing Programs From Specifications. North Holland, 1991

Partial Relations for Program Derivation
Adequacy, Inevitability and Expressiveness

A. M. Haeberer and P. A. S.Veloso

Depto. Informática, Pont. Univ. Católica do Rio de Janeiro
Rua Mq. de S. Vicente 225; Rio de Janeiro, RJ 22453; Brasil

Abstract

A framework for specifying, and reasoning about, problems and programs is presented. This framework is based on partial binary relations (relations together with two carriers sets) and underlies the semantics of any formalism for reasoning about programs together with preconditions. An algebraic calculus is developed and argued to be adequate for program derivation by means of formal manipulation of expressions. Its expressive power is shown to encompass that of first-order logic, thereby solving Tarski's problem on the expressiveness of his calculus of binary relations. Although our problem-theoretic formalism is based on partial binary relations, our considerations are perfectly general.

1. INTRODUCTION

This paper presents a framework, based on *partial binary relations* (relations together with two carriers sets), for specifying, and reasoning about, problems and programs. This framework underlies the semantics of any formalism for reasoning about programs together with preconditions. Based on our formalism we develop a calculus and we argue that its algebraic flavor is adequate for program derivation by means of formal manipulation of expressions. Furthermore, we prove that its expressive power encompasses that of first-order logic, thereby solving the problem posed by Tarski on the expressiveness of his calculus [Tar41] by extending it.

Backhouse et al. [Bac90] advocate the use of a relational calculus for program derivation, on the basis of arguments such as the relational nature, shown by de Moor, of optimization problems without unique optima. Other relational approaches are the calculus for recursive program schemes of de Bakker and de Roever [deB72], as well as the weakest prespecification calculus of Hoare and Jifeng [Hoa86]. Another motivation for a relational approach stems from the formalization by means of partial relations by P. Veloso, S. Veloso, Haeberer, Baum and Elustondo [Vel81, 84; Hae87, 89, 90a] of some of Polya's ideas on problems and problem-solving [Pol57]. A relational approach to programming is employed also by D. Smith [Smi83]. Furthermore, we should mention that the extra flexibility afforded by relations, in contrast to functions, can be very helpful in specifying problems and problem-solving strategies, as well as in manipulating them.

Thus, we have two relational approaches to program derivation. One is based on Tarski's theory of binary relations, whereas the intuitive motivation for the other one comes from both Polya's and Tarski's views. Since we wish to compare them, we start by reviewing Tarski's elementary theory and calculus of binary relations in Section 2, where we also examine the algebraic structure of relational algebras.

In attempting to express something as simple as the palindrome problem, we feel the need to enrich Tarski's repertoire of operations on relations. Further extensions are needed in order to derive a program for this problem. In Section 3 we illustrate how programs for this problem can be derived in our extended formalism. We also illustrate the use of the formalism for expressing eurekas, as well as derivation strategies, such as divide-and-conquer and Dijkstra's decomposition induced by preconditions.

In Section 4 we analyze some implications of our extension to Tarski's calculus. We now have partial relations, in that we can incorporate preconditions into them. We prove that the relational semantics underlying any calculus rich enough for associating preconditions to relations must go beyond Tarski's relations, by involving partial relations.

The main lines of the set-theoretical development of the algebraic theory of problems, which underlies our formalism, are presented in Section 5, where we actually take into account the partiality of the relations.

Implications of our enrichment of Tarski's calculus are examined in Section 6. We begin by discussing the problem posed by Tarski on the expressiveness of his calculus with respect to his elementary theory of relations [Tar41]. We then prove that the expressive power of our calculus encompasses that of first-order logic, by showing how to construct a problem-theoretic version of Tarski's definition of satisfaction.

In Section 7 we establish monotonicity and continuity of our algorithmic operations with respect to the relation of subproblem and present conditions for the monotonicity with respect to refinement.

Section 8 examines some aspects of our problem-theoretic formalism for specifying and constructing problems and programs. In particular, we discuss the meaning of recursive expressions and indicate how to express and manipulate strategies (including their termination conditions) as well as design decisions.

The central results are in Sections 4 and 6 (Theorems 4.4, 4.7, 6.3 and 6.4), whereas Sections 3 and 8 address methodological issues in program derivation.

2. TARSKI'S THEORIES OF RELATIONS

In this Section we will introduce Tarski's *Elementary Theory of Binary Relations* and *Calculus of Binary Relations* [Tar41], analyzing then the algebraic structure of the latter .

2. 1. The Elementary Theory of Relations

Tarski develops this theory as an extension of first-order logic by introducing variables ranging over two sorts, i.e., *individuals*, which will be denoted here by x, y, z, ... , and *relations*, which will be denoted by italic letters r, s, t, ... The atomic sentences are of the form r(x, y) (or x r y -we will use both notations indistinctly - meaning "x is in *relation* r with y") and $r = s$ (where the symbol = denotes equality on *relations*). As usual, the compound sentences are obtained from atomic ones by means of logical connectives $\wedge$, $\vee$, $\leftrightarrow$, $\rightarrow$, $\neg$, and the quantifiers $\forall$ and $\exists$.

The symbols introduced by Tarski are, in our notation, ∞ (for the universal relation), 0 (for the null relation), 1 (for the identity relation) and $\wp$ (for the diversity relation), as relational constants, together with the following operations on *relations* $^{-}$ (complement), $^{\sim}$ (converse), + (sum), • (intersection), $\oplus$ (relative sum), and ; (relative product). The symbols ∞, 0, $^{-}$, + and • are called *absolute* or *Boolean*, whereas 1, $\wp$, $^{\sim}$, $\oplus$ and ;, are called *relative* or *Peircean* (because Peirce developed an early theory of *relations*, later systematized and extended by Schröder [Sch95]).

Finally, Tarski takes as extralogical axioms:

At 1.1 $(\forall x)(\forall y)(\infty(x, y))$

At 1.2 $(\forall x)(\forall y)(\neg\, 0(x, y))$

At 1.3 $(\forall x)(1(x, x))$

At 1.4 $(\forall x)(\forall y)(\forall z)((r(x, y) \wedge 1(y, z) \rightarrow r(x, z)))$

At 1.5 $(\forall x)(\forall y)(\wp(x, y) \leftrightarrow \neg\, 1(x, y))$

At 1.6 $(\forall x)(\forall y)(\bar{r}(x, y) \leftrightarrow \neg\, r(x, y))$

At 1.7 $(\forall x)(\forall y)(\breve{r}(x, y) \leftrightarrow r(y, x))$

At 1.8 $(\forall x)(\forall y)(r + s(x, y) \leftrightarrow (r(x, y) \vee s(x, y)))$

At 1.9 $(\forall x)(\forall y)(r \bullet s(x, y) \leftrightarrow (r(x, y) \wedge s(x, y)))$

At 1.10 $(\forall x)(\forall y)(r \oplus s(x, y) \leftrightarrow (\forall z)(r(x, z) \vee s(z, y)))$

At 1.11 $(\forall x)(\forall y)(r \,;\, s(x, y) \leftrightarrow (\exists z)(r(x, z) \wedge s(z, y)))$

At 1.12 $r = s \leftrightarrow (\forall x)(\forall y)(r(x, y) \leftrightarrow s(x, y))$

2. 2. The Calculus of Binary Relations

To develop this *calculus* Tarski derives from the axioms of the *Elementary Theory of Binary Relations* an appropriate set of theorems whose variables are exclusively *relational* ones. Then, he takes these theorems as the axioms of his *Calculus of Binary Relations*. His extralogical axioms can be divided into three groups:

Axioms of the *absolute* or *Boolean* symbols:

At 2.1 $r + s = s + r$

At 2.2 $r \bullet s = s \bullet r$

At 2.3 $(r + s) \bullet t = (r \bullet t) + (s \bullet t)$

At 2.4 $(r \bullet s) + t = (r + t) \bullet (s + t)$

At 2.5 $r + 0 = r$

At 2.6 $r \bullet \infty = r$

At 2.7 $r + \bar{r} = \infty$

At 2.8 $r \bullet \bar{r} = 0$

At 2.9 $\bar{\infty} = 0$

Axioms of the *relative* or *Peircean* symbols:

At 2.10 $\breve{\breve{r}} = r$

At 2.11 $\widetilde{r \,;\, s} = \breve{s} \,;\, \breve{r}$

At 2.12 $r \,;\, (s \,;\, t) = (r \,;\, s) \,;\, t$

At 2.13 $r \,;\, 1 = r$

Axioms relating *absolute* and *relative* symbols:

At 2.14* $r \,;\, \infty = \infty \vee \infty \,;\, \bar{r} = \infty$

At 2.15 $(r \,;\, s) \bullet \breve{t} = 0 \rightarrow (s \,;\, t) \bullet \breve{r} = 0$

At 2.16 $\wp = \bar{1}$

At 2.17 $r \oplus s = \overline{\overline{r} ; \overline{s}}$

Shorter axiomatizations have been provided, for instance, by Jónsson and Tarski [Jón52] and Chin and Tarski [Chi50]. We adopt the above one in view of our interest in expressiveness (see Section 6). Some of these results (marked with *) were later found too restrictive - in that they hold only for special classes of structures - and then dropped. This is the case of **At** 2.14*, as well as of Theorem 2.14* below. Our development will not rely on them.

The following are examples of the theorems of the *Calculus of Binary Relations* derived by Tarski:

Theorem 2.1 $((r ; s) \bullet t = 0$ iff $(s ; \tilde{t}) \bullet \tilde{r} = 0)$ and
$((r ; s) \bullet t = 0$ iff $(\tilde{t} ; r) \bullet \tilde{s} = 0)$

Theorem 2.2 If $r \bullet \overline{s} = 0$ then $\overline{\tilde{r}} \bullet \tilde{s} = 0$

Theorem 2.3 $\widetilde{r + s} = \tilde{s} + \tilde{r}$

Theorem 2.4 $\tilde{0} = 0$ and $\tilde{\infty} = \infty$

Theorem 2.5 If $s \bullet \overline{t} = 0$ then $(r ; s) \bullet \overline{r ; t} = 0$

Theorem 2.6 $r ; (s + t) = (r ; s) + (r ; t)$

Theorem 2.7 $r ; 0 = 0$

Theorem 2.8 If $r \bullet \overline{s} = 0$ then $(r ; t) \bullet \overline{s ; t} = 0$

Theorem 2.9 $(r + s) ; t = (r ; t) + (s ; t)$

Theorem 2.10 $0 ; r = 0$

Theorem 2.11 $\tilde{1} = 1$

Theorem 2.12 $1 ; r = r$

Theorem 2.13 If $(\infty ; s) \bullet t = 0$ then $(\infty ; t) \bullet s = 0$

Theorem 2.14* If $r \neq \infty$ then $(\infty ; \overline{r}) ; \infty = \infty$

2. 3. The Algebraic Structure of the Algebra of Binary Relations

The intended standard models of Tarski's *calculus* consist of *binary relations* over a *universe* $\mathcal{U}$. Let us consider such a set $\mathfrak{R}_{\mathcal{U}}$ of *binary relations* and analyze its algebraic structure under the relation $\subseteq$ (where, as usual, $r \subseteq s$ means $r + s = s$).

It is easy to observe that:

i. $\subseteq$ is a *partial order*, i.e., $\mathfrak{R}_{\mathcal{U}}$ is a *poset*;

ii. for every pair of *relations* r and s in $\mathfrak{R}_{\mathcal{U}}$, the *relation* $r + s$ (respectively $r \bullet s$) is the *least upper bound* (respectively *greatest lower bound*) of r and s.

Hence, $\langle \mathfrak{R}_{\mathcal{U}}, \subseteq \rangle$ is a *lattice* [Bur80, Grä71]. Thus, both + and • are associative, idempotent and satisfy the absorption laws $r = r + (s \bullet t)$ and $r = r \bullet (s + t)$. It is also easy to notice that $\infty = lub(\mathfrak{R}_{\mathcal{U}})$ and $0 = glb(\mathfrak{R}_{\mathcal{U}})$. In addition, for every $A \subseteq \mathfrak{R}_{\mathcal{U}}$, $lub(A) \in \mathfrak{R}_{\mathcal{U}}$ and $glb(A) \in \mathfrak{R}_{\mathcal{U}}$, hence $\langle \mathfrak{R}_{\mathcal{U}}, \subseteq \rangle$ is a *complete lattice*.

From axioms **At** 2.3 and **At** 2.4 the *lattice* $\langle \mathfrak{R}_{\mathcal{U}}, \subseteq \rangle$ is *distributive*. Then, by a well known theorem of *lattice theory* [Grä71], if there exists a *complement*, it will be unique. Now, since $\langle \mathfrak{R}_{\mathcal{U}}, \subseteq \rangle$ is a *distributive lattice*, axioms **At** 2.5 through **At** 2.8 imply that $\langle \mathfrak{R}_{\mathcal{U}}, \subseteq \rangle$ is a *Boolean Algebra.*

It is quite clear that the *relations* of the form $a = \{\langle x, y\rangle\}$ are the *atoms*, i.e., the only elements $r \in \mathfrak{R}_{\mathcal{U}}$ that satisfy $r \subseteq a$ are 0 and a. Hence, $\langle \mathfrak{R}_{\mathcal{U}}, \subseteq \rangle$ is a *Complete Atomistic Boolean Algebra* [McK40], in that for every $r \neq 0$ there exists an atom a such that $r \subseteq a$.

3. PROGRAM DERIVATION WITH BINARY RELATIONS

In this Section we will extend Tarski's *Calculus of Binary Relations* by means of some new operations and constants required to obtain an appropriate *relational calculus for program construction*. We will do this in an informal and naive way while deriving a well-known programming problem, namely the *Palindrome* problem. The *Palindrome* problem is the problem of checking *whether a given sequence is a palindrome, i.e. whether it reads the same forward and backward* [Par90]. This problem can be expressed by the *relation* $pal = \{\langle x, y\rangle : y = \mathbf{T} \leftrightarrow x = x\sim\}$, where x~ denotes the reversal of x.

Since our problem is about lists, let us structure the domain of lists of elements from a given set C. If we denote by Λ the null list, we can denote the domain of lists by $\mathcal{L}^* = \bigcup_{i \in N} \mathcal{L}^i$, where the $\mathcal{L}^i$'s are the sets of all lists of length i and $\mathcal{L}^0 = \{\Lambda\}$.

Now, we will try to express the palindrome problem by means of a term of Tarski's *Calculus of Binary Relations*. By its very definition, the problem can be restated as:

to check whether a given sequence equals its reversal. (0)

Unfortunately, at this point we already see that the language of Tarski's *Calculus of Binary Relations* does not seem to suffice to capture our specification. Because, since it does not have individual variables, we simply cannot make two copies of a list, which is what appears to be needed to express restatement (0) of the problem.

Hence, we introduce two new operations on *relations*, namely *fork* and *direct product*. For the sake of clarity, let us do it informally.

We call *fork* of *relations* p and q the relation:

$$p \nabla q = \{\langle x, y * z\rangle : \langle x, y\rangle \in p \wedge \langle x, z\rangle \in q\}$$

Notice that the definition of fork involves a new symbol, namely the operation $*$. This operation indicates pair formation: $x * y = [x, y]$. The introduction of this pair-constructing operation has some important consequences. We started with a base set $B = \mathcal{C} \cup \mathcal{L}^*$, consisting of elements (atoms) and lists. We now close this set under the operation $*$, by including *trees* of elements of B. This gives us our universe $\mathcal{U}$. More precisely, the universe $\mathcal{U} = B^*$ is the *free groupoid* generated by the base set $B = \mathcal{C} \cup \mathcal{L}^*$ (its operation being denoted by $*$). Notice that operation $*$ is non-associative (that is why we talk of *trees*, rather than *strings*). In this sense, we now have a structured universe, rather than a mere set of *points*.

In particular, recalling that 1 is the identity on $\mathcal{U}$, we let $2 = 1 \nabla 1$, $3 = 1 \nabla 2$, $4 = 1 \nabla 3$, and so forth. So, 2 is the *relation* $\{\langle u, u * u\rangle : u \in \mathcal{U}\}$ and $3 = \{\langle u, u * (u * u)\rangle : u \in \mathcal{U}\}$.

We define the *direct product* of *relations* p and q as:

$$p \times q = \{\langle x * y, z * t\rangle : \langle x, z\rangle \in p \wedge \langle y, t\rangle \in q\}$$

where $*$ is the above structuring operation on the universe $\mathcal{U} = B^*$.

In order to simplify the notation, we employ $p \times q \times r$ for $p \times (q \times r)$ and similarly for *fork*.

Now, we are able to express (0) as follows:

$$pal = 2 \,;\, (1 \times rev) \,;\, eql \qquad (1)$$

where $rev = \{\langle x, y\rangle : x \in \mathcal{L}^* \wedge y = x{\sim}\}$ is the reversal *relation* on lists and $eql = \{\langle x * y, z\rangle : x \in \mathcal{L}^* \wedge y \in \mathcal{L}^* \wedge z = \mathbf{T} \leftrightarrow x = y\}$ is the *Boolean* equality on lists. Notice that these expressions are not within our calculus.

One way to see that (1) actually expresses (0) is by means of the annotated *Begriffsschrift*-like diagram in figure 3.1 below:

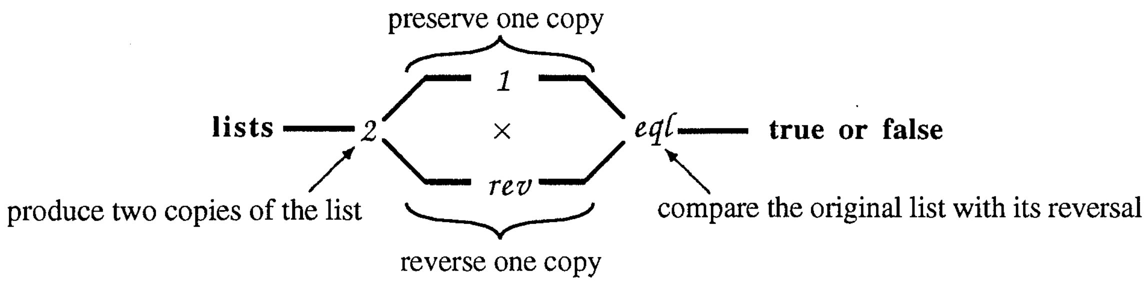

preserve one copy
1
lists
2
×
eql
true or false
rev
produce two copies of the list
compare the original list with its reversal
reverse one copy

Figure. 3.1 - Annotated *Begriffsschrift*-like diagram for the *palindrome* problem

We should express *rev*, *eql*, and also *pal*, not only within our calculus, but also in terms of basic operations on lists.

So, let us also present in a relational manner some usual operations on lists usually built into a programming language

$hd = \{\langle x, c\rangle : x \in \mathcal{L}^* - \mathcal{L}^0 \wedge c \in \mathcal{C} \wedge c = \mathbf{head}(x)\}$
$tl = \{\langle x, y\rangle : x \in \mathcal{L}^* - \mathcal{L}^0 \wedge y = \mathbf{tail}(x)\}$
$cns = \{\langle [c, x], z\rangle : c \in \mathcal{C} \wedge x \in \mathcal{L}^* \wedge z = \mathbf{cons}(c, x)\}$
$init = \{\langle x, y\rangle : x \in \mathcal{L}^* - \mathcal{L}^0 \wedge y = \mathbf{initial}(x)\}$
$lst = \{\langle x, c\rangle : x \in \mathcal{L}^* - \mathcal{L}^0 \wedge c \in \mathcal{C} \wedge c = \mathbf{last}(x)\}$
$app = \{\langle [x, c], y\rangle : x \in \mathcal{L}^* \wedge c \in \mathcal{C} \wedge y = \mathbf{append}(x, c)\}$
$lhd = \{\langle x, y\rangle : x \in \mathcal{L}^* - \mathcal{L}^0 \wedge y = \mathbf{cons}(\mathbf{head}(x), \Lambda)\}$
$llst = \{\langle x, y\rangle : x \in \mathcal{L}^* - \mathcal{L}^0 \wedge y = \mathbf{cons}(\mathbf{last}(x), \Lambda)\}$
$md = \langle x, y\rangle : x \in \mathcal{L}^* - \mathcal{L}^1 \wedge y = \mathbf{initial}(\mathbf{tail}(x))\}$

Let us now derive an expression for the *relation* *rev* (the reversal). Clearly, the reversal of a list of length 0 (i.e., the empty list) or 1 is the list itself. So, we may consider decomposing *rev* into two cases by partitioning its domain into $\mathcal{L}^1$ and $\mathcal{L}^* - \mathcal{L}^1$. If we succeed in doing this, then a relational sum of both cases will give us *rev.* Unfortunately, such domain partitioning is not expressible in Tarski's *Calculus of Binary Relations*, even with the extensions we have just introduced.

So, we extend the theory even further. In Tarski's *Calculus*, we have the constant *1* for the *identity relation* on $\mathcal{U}$. Let us denote the restriction of this identity to set $\mathcal{L}^n$ by $1_{\mathcal{L}^n}$ and similarly for $1_{\mathcal{L}^*}$. We should stress that this is not a mere notational trick; it does increase the expressiveness of Tarski's *Calculus*, since we can now name a *set* by means of a special relation. What we are introducing here is *relativization.* Note that $1_{\mathcal{L}^*} ; rev = rev$ because $\mathcal{D}om(rev) = 1_{\mathcal{L}^*}$, but $1_{\mathcal{L}^n} ; rev \neq rev$. By the definition of $;$, $1_{\mathcal{L}^n} ; rev$ means the restriction of *rev* to $1_{\mathcal{L}^n}$, i.e., $1_{\mathcal{L}^n} ; rev \subset rev$, with $\mathcal{D}om(1_{\mathcal{L}^n} ; rev) = \mathcal{L}^n$. In Section 4 we will take a closer look at this extension.

So, the decomposition of *rev* into two cases by partitioning its domain into $\mathcal{L}^1$ and $\mathcal{L}^* - \mathcal{L}^1$ can be expressed by the equation:

$$1_{\mathcal{L}^*} ; rev = 1_{\mathcal{L}^1} ; rev + 1_{\mathcal{L}^*-\mathcal{L}^1} ; rev \qquad (2)$$

This corresponds to the usual derivation heuristics of *trivialization*. The rationale for this decomposition was the realization of the fact that the reversal of any empty or unitary list is the list itself. This is expressed as $1_{\mathcal{L}^1} \,;\, rev = 1_{\mathcal{L}^1}$. Thus:

$$rev = 1_{\mathcal{L}^1} + 1_{\mathcal{L}^*-\mathcal{L}^1} \,;\, rev \tag{3}$$

So, we are left with the problem of reversing a non-trivial list, that is $rev_1 = 1_{\mathcal{L}^*-\mathcal{L}^1} \,;\, rev$. Now, this is probably a good point to introduce a *eureka*. We can try to take advantage of the inductive structure of the domain $\mathcal{L}^*$. But, instead of tackling an inductive solution in a *divide-and-conquer* fashion, we can actually derive it by reasoning as follows.

We can imagine the last "step" of our *program* as a sort of *join*, such as the concatenation of the ℓhd, the middle part and the ℓst of a decomposed list whose reversed parts we already have. What we have in mind here is a recursive solution, since ℓhd and ℓst are lists of length 1, which are *palindromes* by definition, and the middle part is shorter than the original list. So the *eureka* should be something like:

$$erk = \widetilde{3 \,;\, (\ell hd \times md \times \ell st)} \tag{4}$$

Hence, we can write the following equation:

$$x \,;\, erk = rev_1 \tag{5}$$

The idea behind (5) is that we are looking for a *value* of x which, *multiplied* on its right by erk, yields rev_1. Now, by multiplying both sides of (5) by the *converse* of erk, we have:

$$x \,;\, erk \,;\, \widetilde{erk} = rev_1 \,;\, \widetilde{erk} \tag{6}$$

Since $\mathcal{R}an(\widetilde{erk}) = \mathcal{D}om(erk)$ and $\widetilde{erk}$ is functional, we have $erk \,;\, \widetilde{erk} = 1_{\mathcal{D}om(erk)}$. Then, $x = rev_1 ; \widetilde{erk}$ is a solution for (5), which, in view of (4), we can write as:

$$x = 1_{\mathcal{L}^*-\mathcal{L}^1} \,;\, rev \,;\, 3 \,;\, (\ell hd \times md \times \ell st) \tag{7}$$

At this point, we can apply a property of *forks*: $t \,;\, \mathbb{n} = \mathbb{n} \,;\, t^{\times n}$ (here $t^{\times n}$ means the n-fold *direct product* of t by itself, and $\mathbb{n}$ is the nth *fork*) whenever $t \,;\, \tilde{t} \,;\, t = t$ (which is an algebraic way of stating that t is *deterministic*, i.e., *functional*). Thus, we can replace in (7), $rev \,;\, 3$ by $3 \,;\, (rev \times rev \times rev)$, to obtain:

$$x = 1_{\mathcal{L}^*-\mathcal{L}^1} \,;\, 3 \,;\, (rev \times rev \times rev) \,;\, (\ell hd \times md \times \ell st) \tag{8}$$

By applying a distributivity result on (8), we have:

$$x = 1_{\mathcal{L}^*-\mathcal{L}^1} ; 3 ; ((rev ; \mathit{lhd}) \times (rev ; md) \times (rev ; \mathit{llst})) \qquad (9)$$

But we have $rev ; \mathit{lhd} = \mathit{llst}$ and $rev ; \mathit{llst} = \mathit{lhd}$, as well as $rev ; md = md ; rev$. So, we can rewrite (9) as:

$$x = 1_{\mathcal{L}^*-\mathcal{L}^1} ; 3 ; (\mathit{llst} \times (md ; rev) \times \mathit{lhd})$$

By *unfolding* into (5), we have:

$$rev_1 = 1_{\mathcal{L}^*-\mathcal{L}^1} ; 3 ; (\mathit{llst} \times (md ; rev) \times \mathit{lhd}) ; erk \qquad (10)$$

Finally, by *unfolding* (10) into (3), we obtain:

$$rev = 1_{\mathcal{L}^1} + 1_{\mathcal{L}^*-\mathcal{L}^1} ; 3 ; (\mathit{llst} \times (md ; rev) \times \mathit{lhd}) ; erk \qquad (11)$$

This is a final expression for *rev.*

Now, by definition, whenever $x \in \mathcal{L}^{n+1}$ with $n \geq 1$:

if $\langle x, z \rangle \in md$ then, $z \in \mathcal{L}^{n-1} \subseteq \mathcal{L}^n$,

if $\langle x, z \rangle \in \mathit{lhd}$ then, $z \in \mathcal{L}^1 \subseteq \mathcal{L}^n$, and

if $\langle x, z \rangle \in \mathit{llst}$ then, $z \in \mathcal{L}^1 \subseteq \mathcal{L}^n$.

Thus, equation (11), together with the fact that successive applications of md eventually reduce any list to one with length at most 1, expresses in our formalism the following inductive argument:

Basis:

If $x \in \mathcal{L}^1$ then the reversal of x is x itself.

Inductive step:

If $x \in \mathcal{L}^{n+1}$, with $n \geq 1$, then we have two possible cases:

i) $x \in \mathcal{L}^n$, then, the problem is solved by inductive hypothesis,

ii) $x \notin \mathcal{L}^n$, then, we can:

decompose x into three lists: **cons**(**head**(x), Λ), **cons**(**last**(x), Λ), **initial**(**tail**(x)), each of which belonging to $\mathcal{L}^n$, obtain their reversals (by inductive hypothesis) and join appropriately these lists.

We still have some further comments about the derivation we have developed. After introducing the *eureka*, the whole derivation of a *term* for *rev* is mere calculation. Also, the requirement of equality in equation (5) is usually too strong. What we should require is that $x\ ;\ erk \subseteq rev_1$ and $\mathcal{D}om(x\ ;\ erk) = \mathcal{D}om(rev_1)$. This relation will be denoted by $\leftarrow\!\bullet$ and will be discused formally later on. Here, it suffices to point out that one solution of the equation $x\ ;\ erk \leftarrow\!\bullet\ rev_1$, turns out to be the *weakest pre-specification* introduced by Hoare and Jifeng [Hoa86]. Our solution for $x\ ;\ erk = rev_1$ amounts to their *strongest pre-specification.*

The *Begriffsschrift*-like diagram for (11) appears in figure 2.

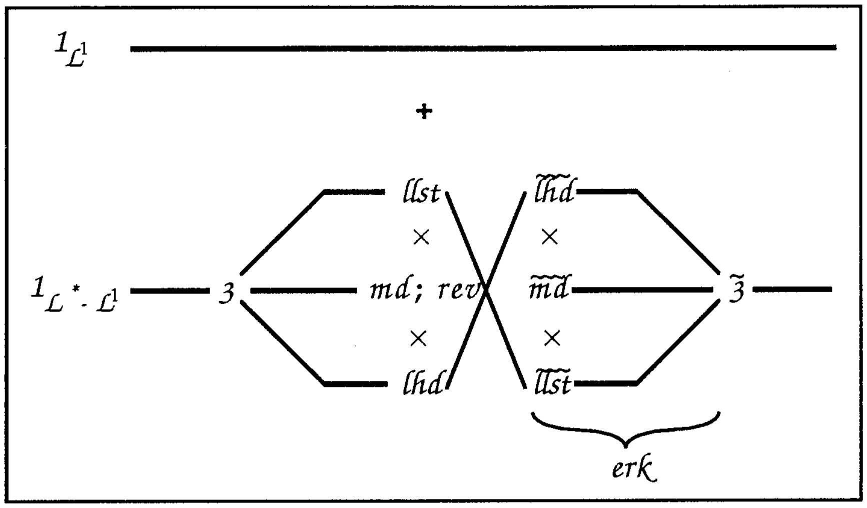


Figure 3. 2 - *Begriffsschrift*-like diagram for *rev*

Notice that the rightmost part of the preceding diagram, i.e. *erk*, can be detached when *rev* is used, as in this case, as part of a larger algorithm.

Now, notice that $\tilde{2} = \{\langle\langle u * v, u\rangle : u \in \mathcal{U} \wedge v \in \mathcal{U} \wedge u = v\rangle\}$. So, by letting

$true = \{\langle u\ , b\rangle : u \in \mathcal{U} \wedge b = \mathbf{T}\}$
$false = \{\langle u\ , b\rangle : u \in \mathcal{U} \wedge b = \mathbf{F}\}$

we can express the specification for equality of lists as

$eql = (1_{\mathcal{L}^*} \times 1_{\mathcal{L}^*})\ ;\ (\tilde{2}\ ;\ true + \tilde{\tilde{2}}\ ;\ false)$

From it, and by similar calculations we can derive an expression for *eql* of the form:

$$eql = trivial + (1_{\mathcal{L}^*-\mathcal{L}^0} \times 1_{\mathcal{L}^*-\mathcal{L}^0})\ ; \quad 2\ ;\ ((hd \times hd) \times (tl \times tl))\ ;\ (smc \times eql)\ ;\ and \tag{12}$$

where:

$$trivial = (1_{\mathcal{L}^0} \times 1_{\mathcal{L}^0}) ; true + (1_{\mathcal{L}^0} \times 1_{\mathcal{L}^*-\mathcal{L}^0}) ; false + (1_{\mathcal{L}^*-\mathcal{L}^0} \times 1_{\mathcal{L}^0}) ; false \qquad (13)$$
$$smc = \{\langle c * d, b\rangle : c \in \mathcal{C} \wedge d \in \mathcal{C} \wedge b = \mathbf{T} \leftrightarrow c = d\}$$
$$and = \{\langle b' * b'', b\rangle : b' \in \text{Bool} \wedge b'' \in \text{Bool} \wedge b = b' \wedge b''\}$$

Note that the use of *trivial* is an example of *filtering* and *smc* is the *Boolean* equality between elements of $\mathcal{C}$. Also, *true* and *false* are *constant relations* (in that each one of them relates any object of the *universe* to a fixed *Boolean* value), whereas *and* is simply the relational counterpart of the conjunction operation on the *sort* Bool = {**T**, **F**}.

Now, equations (1), (11), (4), (12) and (13) form a kind of *equation system*, which characterizes the algorithm we are deriving

$$\begin{cases} pal = 2 ; (1 \times rev) ; eql \\ rev = \underline{1_{\mathcal{L}^1} + 1_{\mathcal{L}^*-\mathcal{L}^1} ; 3} ; (lst \times (md ; rev) \times hd) ; erk \\ erk = 3 ; (hd \times md \times lst) \\ eql = trivial + (1_{\mathcal{L}^*-\mathcal{L}^0} \times 1_{\mathcal{L}^*-\mathcal{L}^0}) ; \\ \qquad\qquad 2 ; ((hd \times hd) \times (tl \times tl)) ; (smc \times eql) ; and \\ trivial = (1_{\mathcal{L}^0} \times 1_{\mathcal{L}^0}) ; true + (1_{\mathcal{L}^0} \times 1_{\mathcal{L}^*-\mathcal{L}^0}) ; false + (1_{\mathcal{L}^*-\mathcal{L}^0} \times 1_{\mathcal{L}^0}) ; false \end{cases}$$

This means that, at the appropriate level of abstraction, this algorithm will

i) solve the palindrome problem by making a copy of the list, reversing it, and comparing it with the original list (1);
ii) reverse the list by, first, splitting it into its head, its middle part and its last, and then interchanging its head and last and recursively reversing its middle part (11);
iii) compare the reversed copy with the original list by recursively splitting both lists into their heads and tails, and comparing, within a kind of join process, their heads as atoms (12).

Note that, if we *unfold* (4) into (11), (13) into (12), then (1), (11) and (12) give an algorithm that *solves* the palindrome problem. But for the sake of efficiency and elegance, and because our goal is to explain how we derive programs by means of this *calculus of relations*, we will continue with our derivation.

We could directly *unfold* (11) into (1). But , in order to simplify the algebraic manipulations, we may try to reuse some decisions used in the derivation of *rev*. In this case, we decide to apply trivialization to *pal*. So,

$$pal = 1_{\mathcal{L}^1} ; 2 ; (1 \times rev) ; eql + 1_{\mathcal{L}^*-\mathcal{L}^1} ; 2 ; (1 \times rev) ; eql$$

By distributing,

$$1_{\mathcal{L}^1} ; 2 ; (1 \times rev) = 2 ; (1_{\mathcal{L}^1} \times 1_{\mathcal{L}^1}) ; (1 \times rev) =$$
$$= 2 ; ((1_{\mathcal{L}^1} ; 1) \times (1_{\mathcal{L}^1} ; rev)) = 2 ; (1_{\mathcal{L}^1} \times (1_{\mathcal{L}^1} ; rev))$$

The trivialization of *rev* gives $1_{\mathcal{L}^1} ; rev = 1_{\mathcal{L}^1}$. Finally,

$$1_{\mathcal{L}^1} ; 2 ; (1 \times rev) = 2 ; (1_{\mathcal{L}^1} \times 1_{\mathcal{L}^1}) = 1_{\mathcal{L}^1} ; 2$$

Similar manipulations on the second summand yield

$$1_{\mathcal{L}^*-\mathcal{L}^1} ; 2 ; (1 \times rev) = 2 ; (1_{\mathcal{L}^*-\mathcal{L}^1} \times (1_{\mathcal{L}^*-\mathcal{L}^1} ; rev))$$

where we can directly unfold (10). Thus we obtain

$$pal = 1_{\mathcal{L}^1} ; 2 ; eql + \qquad (14)$$
$$1_{\mathcal{L}^*-\mathcal{L}^1} ; 2 ; \{1 \times 3 ; (\ell\ell st \times (md ; rev) \times \ell hd) ; erk\} ; eql$$

It is easy to observe that the first summand of (14) can be straightforwardly transformed into $1_{\mathcal{L}^1}$; *true*, which means (∀x)(x ∈ $1_{\mathcal{L}^1}$ → "x is a *palindrome*"). Further manipulations on the second summand, which we will call (b), can begin with an unfolding of *eql* according to (12), and then proceed by simple manipulations, to obtain:

$$\begin{aligned}(b) = \ & 1_{\mathcal{L}^*-\mathcal{L}^1} ; \\ & 2 ; \{1 \times 3 ; (\ell st \times (md ; rev) \times \ell hd) ; erk\} ; \\ & 2 ; ((hd \times hd) \times (tl \times tl)) ; (smc \times eql) ; and\end{aligned} \qquad (15)$$

We have introduced *forks* as a way of producing copies of *individuals*, and then *direct products* to deal with the objects created by *forks*. We now introduce *projections* to decompose such objects into its components. They are convenient shorthand for some special relations, in that they can be defined as follows $\Pi_1 = 1 \nabla \infty$ and $\Pi_2 = \infty \nabla 1$

As usual, *projections* are also useful to rearrange expressions involving *direct products* and *forks*. For instance, $((p \times q) \times (r \times s)) ; t = (p \times r) \times (q \times s)$, where t is the term $(\Pi_1 ; \Pi_1 \nabla \Pi_2 ; \Pi_1) \nabla (\Pi_1 ; \Pi_2 \nabla \Pi_2 ; \Pi_2)$. This illustrates the terms, built with *projections*, *forks* and *direct products*, often used in taking care of the so-called *formal noise* (an expression apparently coined by P. Pepper).

A rearrangement of (15) yields

$$(b) = 1_{\mathcal{L}^*-\mathcal{L}^1} ; 2 ; \{(2 ; (hd \times lst) ; smc) \times (2 ; (tl \times (2 ; ((md ; rev) \times hd) ; app)) ; eql)\} ; and \tag{16}$$

The derivation can now proceed in this manner, namely by algebraic manipulations, without any kind of *eureka* beyond algebraic *gestalt.* These manipulations employ algebraic properties of the various operations, as well as properties of *individuals*, expressed algebraically. For instance, the property of lists: for all $a_1, a_2 \in \mathcal{C} \wedge x_1, x_2 \in \mathcal{L}^*$

$$x_1 \; eql \; x_2 \wedge a_1 \; smc \; a_2 \leftrightarrow cons(a_1, x_1) = cons(a_2, x_2)$$

can be written as:

$$(smc \times eql) ; and = 2 ; t ; (cns \times cns) ; eql$$

where t is the term for taking care of *formal noise* introduced above.

In this manner, one can obtain the expression:

$$pal = 1_{\mathcal{L}^1} ; true + 1_{\mathcal{L}^*-\mathcal{L}^1} ; 2 ; (2 ; (hd \times lst) \times md) ; (smc \times pal) ; and \tag{17}$$

Program (17) can be transformed by simple algebraic manipulations, yielding:

$$pal = 1_{\mathcal{L}^1} ; true + 1_{\mathcal{L}^*-\mathcal{L}^1} ; 2 ; \{((2 ; (hd \times lst) ; smc) \times md\} ; (1_{\mathsf{F}} \times 1) ; \Pi_1 + 1_{\mathcal{L}^*-\mathcal{L}^1} ; 2 ; \{((2 ; (hd \times lst) ; smc) \times md\} ; (1_{\mathsf{T}} \times 1) ; \Pi_2 ; pal \tag{18}$$

which is a clearer instance of the iterative *unary divide-and-conquer scheme* $p = t + q ; p$. Notice that $1_{\mathsf{F}} 1_{\mathsf{T}}$

Both (17) and (18) express executable specifications, i.e. programs. They are executable because they involve only directly executable list operations and algorithmic constructs on relations.

To complete the discussion of the use of the present extension of Tarski's *calculus of relations* in deriving programs from specifications, we will point out some derivation alternatives.

For instance, in deriving an expression for *rev*, we have resorted to a kind of biased *weakest precondition* strategy by means of the introduction of erk as a *eureka* and the equation $x ; erk = 1_{\mathcal{L}^*-\mathcal{L}^1} ; rev$. We may instead follow a somewhat more classical *divide-and-*

conquer strategy, introducing $\widetilde{erk}$ as the *eureka*, and then solving the equation $\widetilde{erk}\ ;\ y = 1_{\mathcal{L}^*-\mathcal{L}^1}\ ;\ rev.$

Also, we have first derived expression (11) for *rev* and then used it in deriving expression (17) for *pal*. We can instead derive (17) directly from (1) by means of properties of *rev* and *eql*. Such a derivation starts by applying the heuristics of *trivialization* directly to (1) to obtain

$$pal = 1_{\mathcal{L}^1}\ ;\ true + 1_{\mathcal{L}^*-\mathcal{L}^1}\ ;\ pal$$

Then, for $1_{\mathcal{L}^*-\mathcal{L}^1}\ ;\ pal$ we apply the *eureka* that any such list can be decomposed into three parts, from which the original list can be recovered. This is expressed by

$1_{\mathcal{L}^*-\mathcal{L}^1} = dcmp\ ;\ rcmb$ with $dcmp = 3\ ;\ (lhd \times md \times llst)$

Thus,

$$1_{\mathcal{L}^*-\mathcal{L}^1}\ ;\ pal = dcmp\ ;\ rcmb\ ;\ 2\ ;\ (1 \times rev)\ ;\ eql.$$

Now, by applying the distributive property of *forks* $t\ ;\ \mathrm{n} = \mathrm{n}\ ;\ t^{\times n}$ and properties of *rev*, we obtain for $1_{\mathcal{L}^*-\mathcal{L}^1}\ ;\ pal$ the expression

$$dcmp\ ;\ 2\ ;\ \{rcmb \times ((\Pi_2\ ;\ \Pi_2)\ \nabla\ ((\Pi_2\ ;\ \Pi_1)\ ;\ rev)\ \nabla\ \Pi_1)\ ;\ rcmb\}\ ;\ eql.$$

If we apply properties of *eql* together with the distributive property of *forks* and rearrange, we can transform this expression into

$$2\ ;\ \{(2\ ;\ (hd \times lst)\ ;\ smc) \times\ 2\ ;\ (md \times (md\ ;\ rev))\ ;\ eql\}\ ;\ and$$

Now *fork* distributivity will give

$$2\ ;\ \{(2\ ;\ (hd \times lst)\ ;\ smc) \times\ (md\ ;\ 2\ ;\ (1 \times rev)\ ;\ eql)\}\ ;\ and$$

whereupon folding of *pal* according to (1) will yield (17).

Section 8 presents some further remarks on these derivations, as well as on derivation strategies in general.

4. TERMINATION AND PARTIALITY

The preceding Section indicates how Tarski's calculus of binary relations can be extended so as to deal with some program derivation needs. The question we start examining in this Section is: do our extensions actually provide a calculus for program derivation?

Let us begin by noticing that the introduction of relativized identities enables us to express, and reason about, partial correctness. Indeed, consider a program denoting relation p and a specification denoted by a relation q and a set $\mathcal{W}$. We can express the partial correctness of the former with respect to the latter by $1_{\mathcal{W}} ; p \subseteq q$.

But, notice that we are dealing with relations over the universe $\mathcal{U}$; thus, they are partial, in the sense of not being defined outside their domains. This means that the corresponding programs fail to terminate outside their domains. We can easily express the domain of a relation r by its identity $1_{\mathcal{D}om(r)} = (r ; \breve{r}) \bullet 1$. But, we cannot so easily calculate with this expression. Also, even if relation r is decidable, its domain may fail to be so, being only partially decidable. These are some of the reasons for using preconditions for specifying decidable subdomains. Several authors, like Dijkstra and Hoare, consider the programmer responsible for providing an adequate precondition.

For a simple-minded example, consider the case of a program whose behavior is the relation $y \geq \frac{a}{x}$ over the reals. Clearly, no one would feed $x = 0$ into this program. So, an appropriate precondition would be, say, $x > 0$. If we call $\mathcal{W}_1 = \{x : x > 0\}$ and $r_1 = \{\langle x, y\rangle : y \geq \frac{a}{x}\}$, then the semantics of the preconditioned program will be the pair $\langle \mathcal{W}_1, r_1\rangle$, termination being expressed by $\mathcal{W}_1 \subseteq \mathcal{D}om(r_1)$.

Generally, one gives a program together with a set of values $\mathcal{R}$ over which it will *terminate*; such a set is usually described by a *precondition*. In our extended version of Tarski's *calculus*, such a set $\mathcal{R}$ can be referred to by means of its identity $1_{\mathcal{R}}$. Clearly, any *Boolean Algebra* $\mathscr{A} = \langle \mathcal{A}, \cup, \cap\rangle$ of sets over $\mathcal{U}$ gives rise to an isomorphic *Boolean Algebra* $\langle \mathfrak{I}_{\mathcal{A}}, +, \bullet\rangle$ of relations, where $\mathfrak{I}_{\mathcal{A}}$ is the set of all the *identity relations* on sets of $\mathcal{A}$. We can restrict a *relation* r to a *precondition* describing a set $\mathcal{W}$ by means of the *relative product* $1_{\mathcal{W}} ; r$ in view of the following simple Lemma.

Lemma 4.1 The restriction of a given *relation* r to a given set $\mathcal{W}$ equals $1_{\mathcal{W}} ; r$.

Proof Since $1_{\mathcal{W}} ; r \subseteq r$, it suffices to prove

$$\mathcal{D}om(1_{\mathcal{W}} ; r) = \mathcal{W} \cap \mathcal{D}om(r) \qquad \text{(i)}$$

In view of the *Boolean Algebra* isomorphism between sets and their identities, (i) is equivalent to $1_{\mathcal{Dom}(1_{\mathcal{W}}\,;\,r)} = 1_{\mathcal{W} \cap \mathcal{Dom}(r)} = 1_{\mathcal{W}} \bullet 1_{\mathcal{Dom}(r)}$.
But $1_{\mathcal{Dom}(1_{\mathcal{W}}\,;\,r)} = 1_{\mathcal{Dom}(1_{\mathcal{W}})} \,;\, 1_{\mathcal{Dom}(r)} = 1_{\mathcal{W}} \,;\, 1_{\mathcal{Dom}(r)}$.
We should note, now, that $1_{\mathcal{W}} \bullet 1_{\mathcal{Dom}(r)} = 1_{\mathcal{W}} \,;\, 1_{\mathcal{Dom}(r)}$.

Q.E.D.

This is not the only way of associating a precondition to a relation. Hoare and Jifeng [Hoa86] represent a set $\mathcal{W}$ by a special relation $c_{\mathcal{W}}$, which they call a condition. A condition $c_{\mathcal{W}}$ has $\mathcal{Dom}(c_{\mathcal{W}}) = \mathcal{W}$ and $\mathcal{Ran}(c_{\mathcal{W}}) = \mathcal{U}$, i. e., $c_{\mathcal{W}} = c_{\mathcal{W}} \,;\, \infty$. (Such conditions, under the name of vectors, are also used by Schmidt and Ströhlein [Sch85] for obtaining representations for relation algebras.) Thus, the restriction of relation r to set $\mathcal{W}$ is represented by $c_{\mathcal{W}} \bullet r$. The connection between these two natural ways of representing sets by binary relations is straightforward, since $c_{\mathcal{W}} = 1_{\mathcal{W}} \,;\, \infty$ and $1_{\mathcal{W}} = c_{\mathcal{W}} \bullet 1$.

So, the effect of imposing a *precondition* denoting $\mathcal{R}$ to the program that computes the *relation* r can be expressed by means of the *relative product* $1_{\mathcal{R}} \,;\, r$. The fact that this program *terminates* over $\mathcal{R}$ is expressed by $\mathcal{R} \subseteq \mathcal{Dom}(r)$ (or, to express it in the *calculus of relations* and not in the *elementary theory of relations*, $1_{\mathcal{R}} \subseteq r \,;\, \breve{r}$). So, we should associate to each *relation* $r \in \mathfrak{R}_{\mathcal{U}}$ a set $\mathcal{R}$, so that the pair $\langle \mathcal{R}, r\rangle$ satisfies the *termination condition* τ: $\mathcal{R} \subseteq \mathcal{Dom}(r)$.

The above considerations suggest that we can express the effect of adding a precondition to a program with our notation. But, is this enough for a program derivation calculus? Assume that we already have a program $\langle \mathcal{W}_1, r_1\rangle$ as above, as well as a program with $r_2 = \{\langle x, y\rangle : y \le (y \le -bx^4 - cx^3 - dx^2 - ex + f\}$ with $\mathcal{W}_2 = \{x : 0 \le x \le h\}$ as its precondition, and that we wish to derive from these a program to find solutions for the system of inequalities

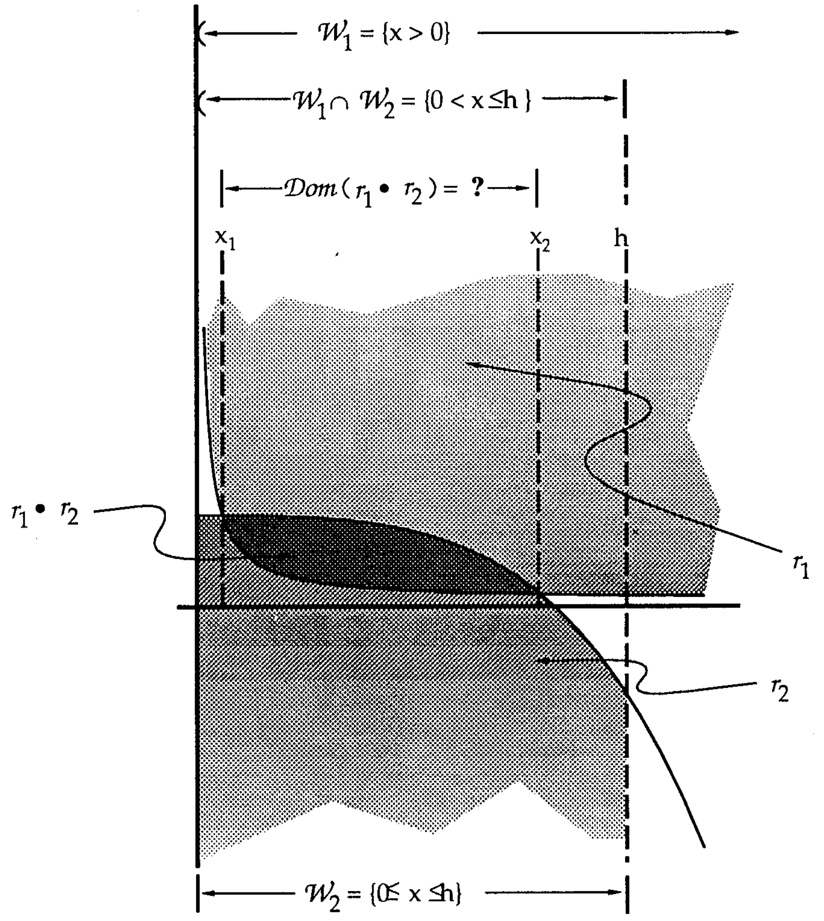
$$\begin{cases} x > 0 \\ y \ge \frac{a}{x} \\ 0 \le x \le h \\ y \le -bx^4 - cx^3 - dx^2 - ex + f \end{cases} \qquad (\varepsilon)$$

We now have the situation depicted in Figure 4.1.

Simple inspection of the this diagram indicates that set $\mathcal{W}_3 = \mathcal{W}_1 \cap \mathcal{W}_2$ does not satisfy the termination condition $\mathcal{W}_3 \subseteq \mathcal{Dom}(r_1 \bullet r_2)$, call it ϖ. In order to obtain a reasonable precondition $\mathcal{W}$ for this program we must determine $\mathcal{Dom}(r_1 \bullet r_2)$; in this case, this amounts to finding points x_1 and x_2 in Figure 4.1. Thus, we have to determine solutions x_1 and x_2 of the system of equations

$$\begin{cases} y = \dfrac{a}{x} \\ y = -bx^4 - cx^3 - dx^2 - ex + f \end{cases}$$

Now, we wish to calculate such a precondition $\mathcal{W}$ from $\langle \mathcal{W}_1, r_1\rangle$ and $\langle \mathcal{W}_2, r_2\rangle$. So, we need an algebraic expression for x_1 and x_2 in terms of the coefficients of the two equations above. But, this is tantamount to having an algebraic expression for the roots of the fifth-degree polynomial $-bx^5 - cx^4 - dx^3 - ex^2 + fx - a$. The latter is known to be impossible since Galois.



$\mathcal{W}_1 = \{x > 0\}$
$\mathcal{W}_1 \cap \mathcal{W}_2 = \{0 < x \leq h\}$
$\mathcal{D}om(r_1 \bullet r_2) = ?$
x_1
x_2
h
$r_1 \bullet r_2$
r_1
r_2
$\mathcal{W}_2 = \{0 \leq x \leq h\}$

Figure 4.1

The point of the above example is twofold. On the one hand, it illustrates that the intersection of preconditions for r_1 and r_2 may fail to be a precondition for $r_1 \bullet r_2$. On the other hand, it indicates that we cannot rely on a calculus of preconditions.

The first point is a consequence of the known property of relations, which is the gist of remark 4.2 below. We have the algebras $\mathscr{A} = \langle \mathcal{A}, \cup, \cap\rangle$, of preconditions, and $\mathscr{R} = \langle \Re_{\mathcal{U}}, +, \bullet\rangle)$, of binary relations. In order to associate to each *relation* r in $\Re_{\mathcal{U}}$ a *precondition* $\mathcal{R}$ in $\mathcal{A}$, we perform the *direct product* of both algebras, which yields a new *Algebra* $\mathscr{A} \times \mathscr{R}$, where each element is an ordered pair $\langle \mathcal{R}, r \rangle$. But what is the effect of restricting these pairs to satisfy the *termination condition* τ? Is the resulting structure still a *Boolean Algebra*? Unfortunately, as will be seen shortly, the answer is no. Let $\mathscr{C} = \{\langle \mathcal{R}, r \rangle \in \mathcal{A} \times \mathfrak{R}_{\mathcal{U}} : \mathcal{R} \subseteq \mathcal{D}om(r)\}$.

Remark 4.2 The set $\mathscr{C}$ of pairs that satisfy the *termination condition* τ is not necessarily closed under $\bullet$.

Proof Given pairs $\langle \mathcal{R}, r \rangle$ and $\langle \mathcal{S}, s \rangle$ in $\mathscr{C}$, we have, by definition:

$\langle \mathcal{R}, r \rangle \bullet \langle \mathcal{S}, s \rangle = \langle \mathcal{R} \cap \mathcal{S}, r \bullet s \rangle$.

If $\langle \mathcal{R} \cap \mathcal{S}, r \bullet s \rangle \in \mathscr{C}$, then $\mathcal{R} \cap \mathcal{S} \subseteq \mathcal{D}om(r \bullet s)$ (i)

But, by assumption,

$\mathcal{R} \subseteq \mathcal{D}om(r)$ and $\mathcal{S} \subseteq \mathcal{D}om(s)$ (ii)

In particular, (ii) will hold with:

$\mathcal{R} = \mathcal{D}om(r)$ and $\mathcal{S} = \mathcal{D}om(s)$ (iii)

Replacing (iii) in (i), we have:

$\mathcal{D}om(r) \cap \mathcal{D}om(s) \subseteq \mathcal{D}om(r \bullet s)$ (iv)

But, by a well known theorem of the theory of *relations* [Sup60] :

$\mathcal{D}om(r \bullet s) \subseteq \mathcal{D}om(r) \cap \mathcal{D}om(s)$ (v)

Thus, (iv) can hold only when $\mathcal{D}om(r) \cap \mathcal{D}om(s) = \mathcal{D}om(r \bullet s)$.

Q.E.D.

In fact, there is a simple necessary and sufficient condition for the *glb* of two terminating preconditioned programs be again so.

Lemma 4.3 Given $\langle \mathcal{R}, r \rangle$ and $\langle \mathcal{S}, s \rangle$ in $\mathscr{C}$, $\langle \mathcal{R}, r \rangle \bullet \langle \mathcal{S}, s \rangle \in \mathscr{C}$ iff $\mathcal{D}om(1_{\mathcal{R}} ; r) \cap \mathcal{D}om(1_{\mathcal{S}} ; s) = \mathcal{D}om((1_{\mathcal{R}} \bullet 1_{\mathcal{S}}) ; (r \bullet s))$.

Proof By assumption,

$\mathcal{Dom}(1_{\mathcal{R}}\,;\,r) = \mathcal{R}$ and $\mathcal{Dom}(1_{\mathcal{S}}\,;\,s) = \mathcal{S}$ (i)

We have, by definition:

$\langle \mathcal{R}, r\rangle \bullet \langle \mathcal{S}, s\rangle = \langle \mathcal{R} \cap \mathcal{S}, r \bullet s\rangle.$

Thus,

$\langle \mathcal{R}, r\rangle \bullet \langle \mathcal{S}, s\rangle \in \mathcal{C}$ iff $\mathcal{R} \cap \mathcal{S} = \mathcal{Dom}((1_{\mathcal{R}} \cap {}_{\mathcal{S}})\,;\,(r \bullet s))$

The latter, by (i), is equivalent to

$\mathcal{Dom}(1_{\mathcal{R}}\,;\,r) \cap \mathcal{Dom}(1_{\mathcal{S}}\,;\,s) = \mathcal{Dom}((1_{\mathcal{R}} \bullet 1_{\mathcal{S}})\,;\,(r \bullet s))$

Q.E.D.

Let us now address the second point illustrated by our example, namely that we cannot rely on a calculus of preconditions. Such a calculus should enable us to derive an appropriate precondition for $r_1 \bullet r_2$ from preconditions $\mathcal{W}_1$ for r_1 and $\mathcal{W}_2$ for r_2, without relying on r_1 and r_2, in particular without computing $\mathcal{Dom}(r_1 \bullet r_2)$.

What Remark 4.2 states is that $\mathcal{C}$ does not form a sublattice of $\langle \mathcal{A}, \cup, \cap\rangle \times \langle \mathfrak{R}_{\mathcal{U}}, +, \bullet\rangle$. But, it might very well be a lattice with its own $\mathcal{glb}$. When can we determine this $\mathcal{glb}$ in a componentwise manner? In fact, does there exist a pair of operations enabling the componentwise calculus? Only in trivial cases in the following sense. Call $\mathcal{C}$ *non-trivial* if it has two pairs that fail to satisfy the condition in Lemma 4.3: $\mathcal{Dom}(1_{\mathcal{R}}\,;\,r) \cap \mathcal{Dom}(1_{\mathcal{S}}\,;\,s) = \mathcal{Dom}((1_{\mathcal{R}} \bullet 1_{\mathcal{S}})\,;\,(r \bullet s))$.

Theorem 4.4 If $\mathcal{C}$ is non-trivial, then there exists no pair of binary operations f, on $\mathcal{A}$, and g, on $\mathfrak{R}_{\mathcal{U}}$, such that $\langle f(\mathcal{R}, \mathcal{S}), g(r, s)\rangle = \mathcal{glb}\{\langle \mathcal{R}, r\rangle, \langle \mathcal{S}, s\rangle\}$, for every $\langle \mathcal{R}, r\rangle$ and $\langle \mathcal{S}, s\rangle$ in $\mathcal{C}$.

Proof Pick $\langle \mathcal{R}, r\rangle$ and $\langle \mathcal{S}, s\rangle$ in $\mathcal{C}$ such that

$\mathcal{Dom}(1_{\mathcal{R}}\,;\,r) \cap \mathcal{Dom}(1_{\mathcal{S}}\,;\,s) \neq \mathcal{Dom}((1_{\mathcal{R}} \bullet 1_{\mathcal{S}})\,;\,(r \bullet s)).$

On the one hand, notice that

$\langle \emptyset, r\rangle$ and $\langle \emptyset, s\rangle$ are in $\mathcal{C}$ and $\mathcal{glb}\{\langle \emptyset, r\rangle, \langle \emptyset, s\rangle\} = \langle \emptyset, r \bullet s\rangle.$

Thus, $g(r, s)\rangle = r \bullet s.$

On the other hand, consider

$\mathcal{R}$ and $\mathcal{S}$ in $\mathcal{A}$ and notice that $\langle \mathcal{R}, \infty \rangle$ and $\langle \mathcal{S}, \infty \rangle$ are in $\mathcal{C}$ and

$glb\{\langle \mathcal{R}, \infty \rangle, \langle \mathcal{S}, \infty \rangle\} = \langle \mathcal{R} \cap \mathcal{S}, \infty \rangle$.

Thus, $f(\mathcal{R}, \mathcal{S}) = \mathcal{R} \cap \mathcal{S}$.

Therefore $\langle f(\mathcal{R}, \mathcal{S}), g(r, s) \rangle = \langle \mathcal{R} \cap \mathcal{S}, r \bullet s \rangle \notin \mathcal{C}$.

Q.E.D.

Notice that naming $\mathcal{C}$ non-trivial if the equality $\mathcal{D}om(1_{\mathcal{R}} ; r) \cap \mathcal{D}om(1_{\mathcal{S}} ; s) = \mathcal{D}om((1_{\mathcal{R}} \bullet 1_{\mathcal{S}}) ; (r \bullet s))$ fails to hold is not a misnomer because enforcing such condition indeed trivializes the structure. Sufficient conditions for non-triviality are, for instance, (i) the existence of two total *relations* with disjoint *ranges*, or (ii) $\mathcal{A}$ being rich enough so as to contain the *domains* of the *relations* in $\mathfrak{R}_{\mathcal{U}}$. Notice that (i) is illustrated by *relations* such as *true* and *false* of Section 3. On the other hand, the non satisfaction of (ii) would preclude those algorithms that fit their *preconditions* like a glove. Thus, any structure $\mathcal{C}$ appropriate for program derivation will indeed be non-trivial.

Theorem 4.4 shows that there is no componentwise manner of determining the glb of terminating preconditioned programs which gives as a result a preconditioned terminating program. In a sense this is due to the behavior of the domain of the glb of relations, as noticed in remark 4.3. A refinement of these ideas indicates that the situation is even worse. If all one knows is $\mathcal{D}om(r)$ and $\mathcal{D}om(s)$, then $\mathcal{D}om(r \bullet s)$ may be just about anything that can be expected, i. e., within $\mathcal{D}om(r) \cap \mathcal{D}om(s)$. This is the content of the next remark.

Remark 4.5 Given any relations r and s and an arbitrary set $\mathcal{W} \subseteq \mathcal{D}om(r) \cap \mathcal{D}om(s)$, let $r' = r ; \infty ; 1_{\mathcal{W}}$ and $s' = 1_{\mathcal{W}} ; s ; \infty ; 1_{\mathcal{W}} + \overline{1_{\mathcal{W}}} ; s ; \infty ; \overline{1_{\mathcal{W}}}$. Then, $\mathcal{D}om(r') = \mathcal{D}om(r)$ and $\mathcal{D}om(s') = \mathcal{D}om(s)$; but $\mathcal{D}om(r' \bullet s') = \mathcal{W}$.

Proof Let $s_1 = 1_{\mathcal{W}} ; s ; \infty ; 1_{\mathcal{W}}$ and $s_2 = \overline{1_{\mathcal{W}}} ; s ; \infty ; \overline{1_{\mathcal{W}}}$.

Then $s' = s_1 + s_2$, and $s_1 \subseteq 1_{\mathcal{W}} ; \infty ; 1_{\mathcal{W}}$ and $s_2 \subseteq \overline{1_{\mathcal{W}}} ; \infty ; \overline{1_{\mathcal{W}}}$.

Similarly, with $r_1 = 1_{\mathcal{W}} ; r ; \infty ; 1_{\mathcal{W}}$ and $r_2 = \overline{1_{\mathcal{W}}} ; r ; \infty ; 1_{\mathcal{W}}$, we

have $r' = r_1 + r_2$, and $r_1 \subseteq 1_{\mathcal{W}} ; \infty ; 1_{\mathcal{W}}$ and $r_2 \subseteq \overline{1_{\mathcal{W}}} ; \infty ; 1_{\mathcal{W}}$.

So, $r_2 \bullet s_2 \subseteq (\infty \,;\, 1_{\mathcal{W}}) \bullet (\infty \,;\, \overline{1_{\mathcal{W}}}) \subseteq \infty \,;\, (1_{\mathcal{W}} \bullet \overline{1_{\mathcal{W}}}) = 0$.

Similarly, $r_2 \bullet s_1 \subseteq (\overline{1_{\mathcal{W}}} \bullet 1_{\mathcal{W}}) \,;\, \infty = 0$ and $r_1 \bullet s_2 = 0$.

Thus, $r' \bullet s' = r_1 \bullet s_1 = 1_{\mathcal{W}} \,;\, [(r \,;\, \infty) \bullet (s \,;\, \infty)] \,;\, 1_{\mathcal{W}}$.

Now, we have

$\mathcal{D}om(s') = \mathcal{D}om(s' \,;\, \infty) =$

$= \mathcal{D}om(1_{\mathcal{W}} \,;\, s \,;\, \infty + \overline{1_{\mathcal{W}}} \,;\, s \,;\, \infty) =$

$= \mathcal{D}om(s \,;\, \infty) = \mathcal{D}om(s)$.

Similarly, $\mathcal{D}om(r') = \mathcal{D}om(r' \,;\, \infty) = \mathcal{D}om(r)$.

Finally, $\mathcal{D}om(r' \bullet s') = \mathcal{D}om(1_{\mathcal{W}} \,;\, [(r \,;\, \infty) \bullet (s \,;\, \infty)]) = \mathcal{W}$.

Q.E.D.

In the preceding remark we obtained the new relation s' from the given s in a manner akin to the strategy of case division. Thus, if we have relations r and s in a derivation, we may very well obtain a relation like $r' \bullet s'$, whose domain bears very little relationship to the domains of r and s.

The preceding results provide an adequate framework for the analysis of the general situation illustrated by our preceding examples. For this purpose, consider Figure 4.2.

The left and right top lattices represent cases as in the example. The left top lattice, τ_λ, depicts the pairs $\langle \mathcal{R}, r \rangle$ such that $\mathcal{R} \subseteq \mathcal{D}om(r)$, which represent all terminating, hence in $\mathscr{C}$, preconditioned programs with behavior r. So, our first program is represented by such a point $\langle \mathcal{R}, r \rangle$. Similarly, the top right lattice, τ_ρ, represents all terminating preconditioned programs with behavior s. So, our second program is represented by such a point $\langle \mathcal{S}, s \rangle$. The bottom lattice $\mathscr{P}$ represents the glb's of elements of τ_λ and τ_ρ. So, $\mathscr{P} = \{\langle \mathcal{R}, r \rangle \bullet \langle \mathcal{S}, s \rangle \in \mathcal{A} \times \mathfrak{R}_{\mathcal{U}} : \langle \mathcal{R}, r \rangle \in \tau_\lambda \wedge \langle \mathcal{S}, s \rangle \in \tau_\rho\}$; now, in $\mathscr{A} \times \mathscr{R}$, we have $\langle \mathcal{R}, r \rangle \bullet \langle \mathcal{S}, s \rangle = \langle \mathcal{R} \cap \mathcal{S}, r \bullet s \rangle$. As such, $\mathscr{P}$ is in one-to-one correspondence with the powerset of $\mathcal{D}om(r \bullet s)$. This lattice $\mathscr{P}$, which represents preconditioned programs with behavior $r \bullet s$, has two interesting parts. The bottom one, τ, represents terminating preconditioned programs with behavior $r \bullet s$, whereas the top one, $\mathscr{C}$, represents programs with behavior $r \bullet s$ whose precondition may be too big in the sense of including $\mathcal{D}om(r \bullet s)$. This is to be expected, even though both $\tau_\lambda \subseteq \mathscr{C}$ and $\tau_\rho \subseteq \mathscr{C}$, in view of remark 4.2.

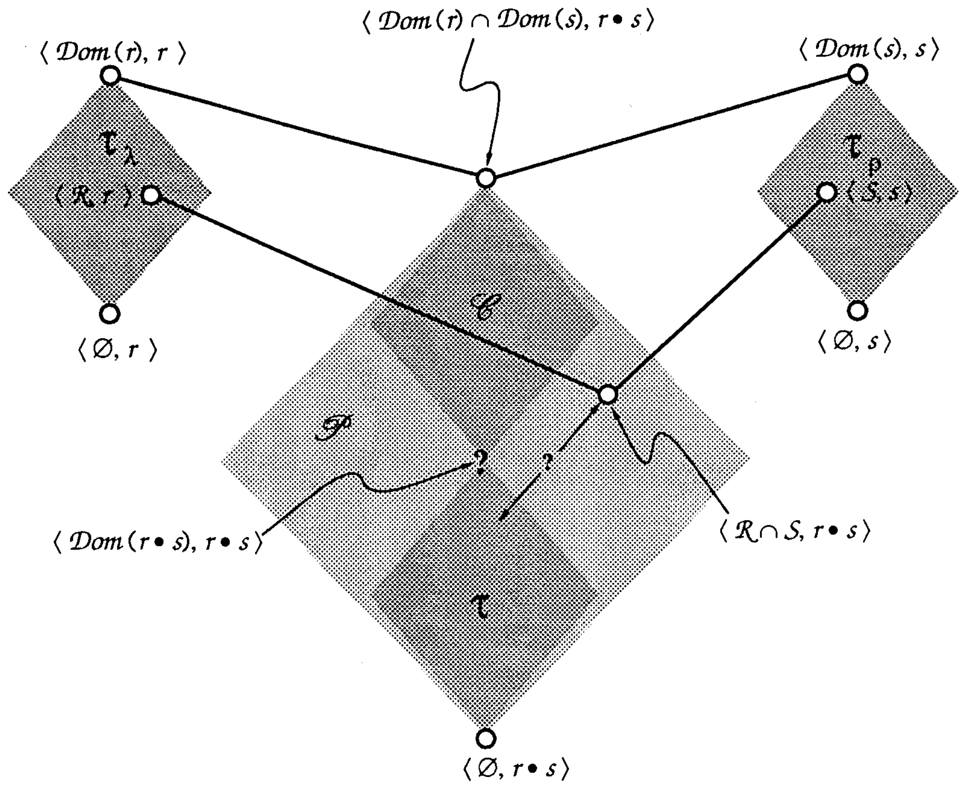


Figure 4.2

Now, we would like to have $\langle \mathcal{R}, r\rangle \bullet \langle \mathcal{S}, s\rangle$ in the bottom part τ. Theorem 4.4 states that we cannot calculate a precondition for $r \bullet s$ solely from $\mathcal{R}$ and $\mathcal{S}$. Now, consider point $\langle \mathcal{D}om(r \bullet s), r \bullet s\rangle$. Remark 4.5 tells us that this point may be anywhere within $\mathcal{P}$. Thus, we cannot actually pin down its position, and hence do not know the boundary between τ and $\mathcal{C}$. Thus, concerning $\langle \mathcal{R}, r\rangle \bullet \langle \mathcal{S}, s\rangle$, we do not know whether it is inside τ, neither do we know how far it is from τ.

One might say that the culprit is the operation of $\mathcal{glb}$ on relations. In fact, some authors [Hoa86] suggest eliminating it (just like the goto) from the programming language, at least. But this suggestion has some undesirable consequences. For, ∇ and $\bullet$ are intimately connected, as indicated in the next lemma.

Lemma 4.6 Given relations r and s, we have

(a) $x \; r \nabla s \; w$ iff $(\exists y)(\exists z)(w = y * z \wedge x \; r \; y \wedge x \; s \; z)$, for all $x, w \in \mathcal{U}$;

(b) $r \bullet s = (r \nabla s) \; ; \widetilde{2}$ and

(c) $r \nabla s = (r \nabla \infty) \bullet (\infty \nabla s) = (r \,;\, \breve{\Pi}_1) \bullet (s \,;\, \breve{\Pi}_2)$.

Proof Immediate from the definitions.

Q.E.D.

Equation (b) in lemma 4.6 implies that any set of binary relations that is closed under ∇ and ; and includes the filter-like equality $\tilde{\mathcal{Z}}$ (which it will if it is closed under converse) must be closed under $\bullet$, as well. Thus, eliminating intersection entails eliminating fork as well. The previous section suggests that fork is a useful tool for expressing programs, in that it enables one to produce copies. (In section 7 we will prove that with fork we have the expressive power of first-order logic, which makes our extension adequate for expressing specifications as well.) On the other hand, part (a) of lemma 4.6 shows that fork can be defined in terms of existential quantifiers and conjunctions. Thus, by eliminating fork, our expressive power will fall short from first-order logic, which will have deleterious effects for a derivation calculus.

Summing up our discussion so far, we can say that there is no calculus of preconditions adequate for program derivation. The only information we can give for such *termination condition* is an *upper bound* for it. If we notice that $\mathcal{D}om(r \bullet s)$ is the largest set that can be involved in a *termination condition* with $r \bullet s$, then it is evident that $(\forall x)(x \in (\mathcal{D}om(r) \cap \mathcal{D}om(s) - \mathcal{D}om(r \bullet s)) \rightarrow (r \bullet s)\uparrow$ (i.e., there will be no pair in $r \bullet s$ whose first element is x). This suggests reversing the inclusion sign in the *termination condition*, i. e., replacing it by $\mathcal{R} \supseteq \mathcal{D}om(r)$. Let us see what we can expect to gain form this suggestion.

Let us examine again the objects $\langle \mathcal{R}, r\rangle$ of the *algebra* $\mathscr{A} \times \mathscr{R}$ with respect to the *termination condition.* The termination condition characterizes two interesting subsets of the *algebra* $\mathscr{A} \times \mathscr{R}$, namely the subset τ of terminating preconditioned programs, and the subset $\mathscr{D} = \{\langle \mathcal{R}, r\rangle : \mathcal{R} \supseteq \mathcal{D}om(r)\}$. We already know that τ is not closed under the *glb* of $\mathscr{A} \times \mathscr{R}$. But, it is easy to see that $\mathscr{D}$ is a better behaved subset of $\mathscr{A} \times \mathscr{R}$.

Theorem 4.7 The subset $\mathscr{D} = \{\langle \mathcal{R}, r\rangle : \mathcal{R} \supseteq \mathcal{D}om(r)\}$ forms a sublattice of $\mathscr{A} \times \mathscr{R}$.

Proof Given pairs $\langle \mathcal{R}, r\rangle$ and $\langle \mathcal{S}, s\rangle$ in $\mathscr{D}$, we have, by definition:

$\langle \mathcal{R}, r\rangle + \langle \mathcal{S}, s\rangle = \langle \mathcal{R} \cup \mathcal{S}, r + s\rangle$ and

$\langle \mathcal{R}, r\rangle \bullet \langle \mathcal{S}, s\rangle = \langle \mathcal{R} \cap \mathcal{S}, r \bullet s\rangle$.

Since $\mathcal{R} \supseteq \mathcal{D}om(r)$ and $\mathcal{S} \supseteq \mathcal{D}om(s)$, we have

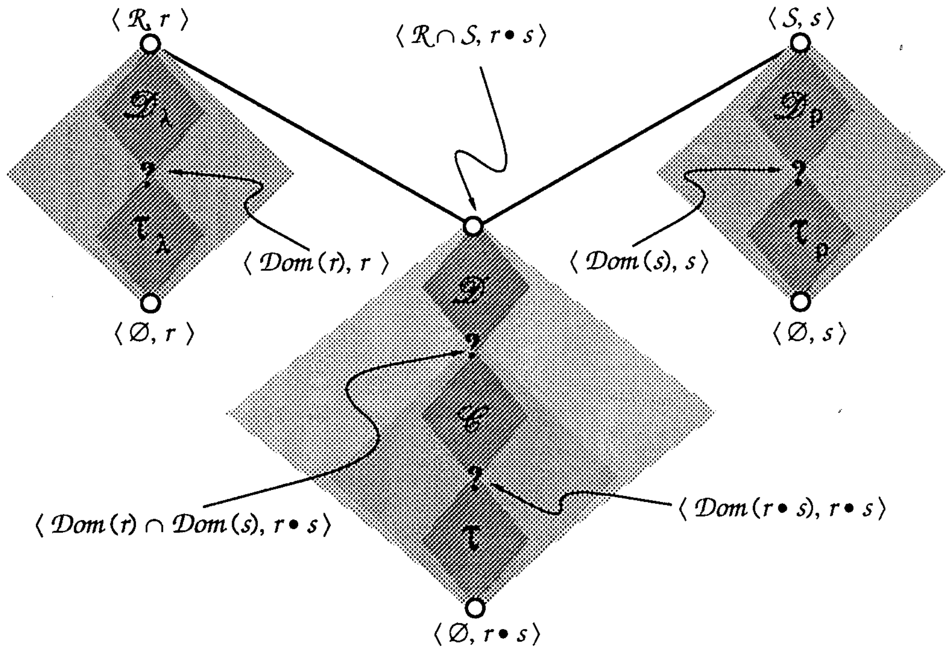
$$\mathcal{R} \cup \mathcal{S} \supseteq \mathcal{D}om(r) \cup \mathcal{D}om(s) = \mathcal{D}om(r + s) \text{ and}$$

$$\mathcal{R} \cap \mathcal{S} \supseteq \mathcal{D}om(r) \cap \mathcal{D}om(s) \supseteq \mathcal{D}om(r \bullet s)$$

Q.E.D.

But, we gain more than simple closure by considering $\mathscr{D}$ instead of τ. This will become clearer from an analysis of Figure 4.3.

Each lattice in Figure 4.3 is similar to the bottom lattice in Figure 4.2, but for the fact that now the top element consists of a relation and a set including its domain. For a given such pair $\langle \mathcal{R}, r\rangle$ with $\mathcal{R} \supseteq \mathcal{D}om(r)$, the left top lattice depicts all pairs $\langle \mathcal{X}, r\rangle$, with the given relation r, with $\mathcal{X} \subseteq \mathcal{R}$. Thus, it represents all preconditioned programs with behavior r and precondition included in $\mathcal{R}$. Similarly to Figure 4.2, the totally correct ones form the bottom part τ_λ, whereas those with termination domain including $\mathcal{D}om(r)$ form the top part $\mathscr{D}_\lambda$. Likewise, the right top lattice depicts all pairs $\langle \mathcal{Y}, s\rangle$, representing preconditioned programs with behavior s whose precondition is upper-bounded by $\mathcal{S}$, and analogously for its parts τ_ρ and $\mathscr{D}_\rho$.



$\langle \mathcal{R}, r \rangle$
$\langle \mathcal{R} \cap \mathcal{S}, r \bullet s \rangle$
$\langle \mathcal{S}, s \rangle$
$\mathscr{D}_\lambda$
?
τ_λ
$\mathscr{D}_\rho$
?
τ_ρ
$\langle \mathcal{D}om(r), r \rangle$
$\langle \mathcal{D}om(s), s \rangle$
$\langle \varnothing, r \rangle$
$\langle \varnothing, s \rangle$
$\mathscr{D}$
?
$\mathscr{C}$
?
τ
$\langle \mathcal{D}om(r) \cap \mathcal{D}om(s), r \bullet s \rangle$
$\langle \mathcal{D}om(r \bullet s), r \bullet s \rangle$
$\langle \varnothing, r \bullet s \rangle$

Figure 4.3

The bottom lattice represents the *glb*'s of elements of the top two lattices. As such, it represents the preconditioned programs with behavior to $r \bullet s$ whose preconditions are included in $\mathcal{R} \cap \mathcal{S}$. This bottom lattice has two interesting, but unfortunately hard to pin down, points, marked as ? in Figure 4.3. The upper ? represents $\langle \mathcal{D}om(\mathcal{R}) \cap \mathcal{D}om(\mathcal{S}), r \bullet s\rangle$ and separates the part $\mathscr{D} = \{\langle \mathcal{X} \cap \mathcal{Y}, r \bullet s\rangle : \mathcal{X} \supseteq \mathcal{D}om(r) \wedge \mathcal{Y} \supseteq \mathcal{D}om(s)\}$ from the bottom lattice $\mathscr{P}$ of Figure 4.3. The lower ? divides $\mathscr{P}$ into two parts τ and $\mathscr{C}$, as before. Now, we already know that, for $\langle \mathcal{X}, r\rangle \in \mathscr{D}_\lambda$ and $\langle \mathcal{Y}, s\rangle \in \mathscr{D}_\rho$, $\langle\langle \mathcal{X}, r\rangle \bullet \langle \mathcal{Y}, s\rangle$ may be just about anywhere within $\mathscr{P}$. On the other hand, for $\langle \mathcal{X}, r\rangle \in \tau_\lambda$ and $\langle \mathcal{Y}, s\rangle \in \tau_\delta$, we do have $\langle\langle \mathcal{X}, r\rangle \bullet \langle \mathcal{Y}, s\rangle$ in $\mathscr{D}$, in view of Theorem 4.7.

Let us now interpret the above remarks in the context of program derivation. The reason for marking some points with ? is the fact that in general we cannot calculate the domain of a relation, thus we have some uncertainty about it. One way to get around this is by means of preconditions, which means that our uncertainty is some lattice like one of the top τ's in Figure 4.3. But then, when continue as in our example towards $r \bullet s$, this information, which was partial, is lost, for our uncertainty is now $\mathscr{P}$. On the other hand, if we work with upper bounds for the domains, our uncertainty is some lattice as one of the $\mathscr{D}$'s in Figure 4.3. Then, it is preserved, in that the uncertainty with respect to $r \bullet s$ is the sublattice $\mathscr{D}$ of the bottom lattice in Figure 4.3.

This discussion clarifies the role of the upper bound on the domain of a relations. If all that one wishes is to reverse the inclusion in the termination condition, one might as well employ a very loose upper bound, say the universe $\mathcal{U}$. We clearly have $\mathcal{D}om(r) \subseteq \mathcal{U}$, but then we lose information. For $\mathcal{D}om(r) \subseteq \mathcal{R}$ gives some information concerning non-termination, namely outside $\mathcal{R}$ we know that r is not defined. In terms of Figure 4.3, the use of $\mathcal{U}$ instead of a tighter upper bound for the domain, would increase the sizes of the lattices $\mathscr{D}$'s , and hence the uncertainty concerning termination domains. Notice that using the universal upper bound $\mathcal{U}$ is equivalent to totalizing all relations by the addition of a bottom element $\perp$.

The net product of our development so far is the conclusion that a *Calculus of Binary Relations* appears to be appropriate for program derivation, but only as a first approximation, since one needs to associate sets to such relations in order to take into account the question of termination.

We wish to develop a *Programming Calculus* based on a *Calculus of Binary Relations*. This calculus will deal with elements of $\mathscr{A} \times \mathscr{R}$. Let us call $\mathbb{P}$ the set of such objects. If we whish to derive as in Section 3, we should make $\mathbb{P}$ closed under *converse*. By this we mean that, if we have in $\mathbb{P}$ an object representing a *partial relation* r on $\mathcal{R}$, i.e., $\mathcal{R} \supseteq \mathcal{D}om(r)$, then we should also have in $\mathbb{P}$ an object corresponding to a *partial relation* $\tilde{r}$ on some $\mathcal{R}^*$, i.e., $\mathcal{R}^* \supseteq \mathcal{D}om(\tilde{r}$). So, the information we should give about our *partial relations* can be presented as a 3-tuple $\langle \mathcal{R}, \mathcal{R}^*, r\rangle$ subject to the restrictions $\mathcal{R} \supseteq \mathcal{D}om(r)$ and $\mathcal{R}^* \supseteq \mathcal{D}om(\tilde{r}$). It easy to see that both conditions can be expressed together as $r \subseteq \mathcal{R} \times \mathcal{R}^*$. Such objects will be called *problems*, since their structure resembles the notion of problem introduced by G. Polya [Pol57]. Polya suggested three questions in approaching a problem: *what are the data* ? *what are* *the results* ? and *what is the problem condition* ? The answers to these questions [Vel84] yield the same structure $\langle \mathcal{R}, \mathcal{R}^*, r\rangle$ we have arrived at.

5. THE ALGEBRAIC THEORY OF PROBLEMS

We shall now start the development of an *Algebraic Theory of Problems* appropriate for program derivation. First of all, we should precisely define our *universe*. Let $\mathbb{S}$ be a class of *basic sorts*. We will consider each basic sort $\mathcal{W}_j$ described by a unary relativization predicate ω_j and denote by B the closure of $\mathbb{S}$ under $\cup$ and $\cap$. Now, the *universe* $\mathcal{U} = B^*$ is the *free groupoid* generated by the base set B.

Definition 5.1. A *problem* over $\mathbb{S}$ is a 3-tuple $\mathbf{P} = \langle D_P, R_P, p\rangle$ where D_P and R_P are subsets of $\mathcal{U}$ and $p \subseteq D_P \times R_P$.

We will refer to the set D_P as the *data carrier*, to the set R_P as the *result carrier* and to the *relation p* as the *condition*, of the *problem* **P**.

We will denote by **1** the *problem* $\langle \mathcal{U}, \mathcal{U}, 1\rangle$. We have $\mathcal{W}_j = \{u : u \in \mathcal{U} \wedge \omega_j(x)\}$, then we define the *problem* $\mathbf{1}_{\mathcal{W}_j} = \langle \mathcal{W}_j, \mathcal{W}_j, 1_{\mathcal{W}_j}\rangle$.

From above discussion the natural ordering on $\mathbb{P}_{\mathcal{U}}$ is "being a *subproblem*", which is defined as follows:

Definition 5.2. $\mathbf{P} \subseteq \mathbf{Q} \leftrightarrow (D_P \subseteq D_Q \wedge R_P \subseteq R_Q \wedge p \subseteq q)$

Definition 5.3. $\mathbf{P} = \mathbf{Q} \leftrightarrow (D_P = D_Q \wedge R_P = R_Q \wedge p = q)$

For some developments it is clearer to present a *problem* **P** as a 3-tuple $\langle P^{\uparrow}, P_{\uparrow}, p\rangle$ where $P^{\uparrow} = R_P - \mathcal{R}an(p)$ and $P_{\uparrow} = D_P - \mathcal{D}om(p)$, whence $P_{\uparrow} \cap \mathcal{D}om(p) = P^{\uparrow} \cap \mathcal{R}an(p) = \varnothing$. We will use the notation:

$$\mathbf{P} = p\left|\frac{P_{\uparrow}}{P^{\uparrow}}\right.$$

which displays a *problem* by means of its *condition* together with two sets $P_{\uparrow}$ (*data indefinition set*) and $P^{\uparrow}$ (*non-reachable result set*).

Recalling the discussion in the previous Section, the relationship between $\mathcal{D}om(p)$ (or $\mathcal{R}an(p)$) and D_P (or R_P) defines some special classes of *problems*. We will say that *problem* **P** is *viable* [Vel84] (which will be denoted $\mathcal{V}i\mathcal{b}(\mathbf{P})$) if its *condition* is total over D_P. We can express formally this definition as $\mathcal{Vib}(\mathbf{P})$ iff $(\forall d)(d \in D_P \rightarrow (\exists r)(r \in R_P \wedge p(d, r)))$, or, in a relational manner as, $\mathcal{Vib}(\mathbf{P})$ iff $1_{D_P} \subseteq p\ ;\ \breve{p}$. Using the alternative representation:

$$\mathcal{Vib}(\mathbf{P}) \leftrightarrow \mathbf{P} = p \left| \frac{\varnothing}{\mathrm{P}^{\uparrow}} \right. \tag{19}$$

A *surjective problem* will be one whose *condition* exhausts its *result carrier*. Formally, $\mathcal{Sur}(\mathbf{P})$ iff $(\forall r)(r \in R_P \rightarrow (\exists d)(d \in D_P \wedge p(d, r)))$, i.e. $1_{D_P} \subseteq \breve{p}\ ;\ p$. Or, equivalently

$$\mathcal{Sur}(\mathbf{P}) \leftrightarrow \mathbf{P} = p \left| \frac{\mathrm{P}_{\uparrow}}{\varnothing} \right. \tag{20}$$

Clearly, Tarski's axioms hold for the *viable-surjective problems*, which will be called *Tarskian problems*, or simply *T-problems*. Formally, $\mathcal{Tar}(\mathbf{P}) \leftrightarrow \mathcal{Sur}(\mathbf{P}) \wedge \mathcal{Vib}(\mathbf{P})$, i.e.,

$$\mathcal{Tar}(\mathbf{P}) \leftrightarrow \mathbf{P} = p \left| \frac{\varnothing}{\varnothing} \right. . \tag{21}$$

Similarly, we call **P** *injective* if its *condition* is *injective*, i.e., $\mathcal{Inj}(\mathbf{P})$ iff $p\ ;\ \breve{p} \subseteq 1$, which obviously can be expressed as $\mathcal{Inj}(\mathbf{P})$ iff $\breve{p}\ ;\ p\ ;\ \breve{p} = \breve{p}$

We call **P** *deterministic* if its *condition* is functional, i.e., $\mathcal{Det}(\mathbf{P})$ iff $\breve{p}\ ;\ p \subseteq 1$, or $\mathcal{Det}(\mathbf{P})$ iff $p\ ;\breve{p}\ ;\ p = p$.

5. 1. The Algebraic Structure of $\langle \mathbb{P}_{\mathcal{U}}, \subseteq \rangle$

It is easy to see that $\subseteq$ is a *partial ordering*. A very important property of any algebraic structure is the existence of a *lub* and a *glb* for each pair of elements in it. Given *problems* **P** and **Q** in $\mathbb{P}_{\mathcal{U}}$:

$$\mathcal{lub}\{\mathbf{P}, \mathbf{Q}\} = \langle D_P \cup D_Q, R_P \cup R_Q, p \cup q \rangle,$$

$$\mathcal{glb}\{\mathbf{P}, \mathbf{Q}\} = p \bullet q \left| \frac{((\mathrm{P}_{\uparrow} \cup \mathcal{Dom}(p)) \cap (\mathrm{Q}_{\uparrow} \cup \mathcal{Dom}(q))) - \mathcal{Dom}(p \bullet q)}{((\mathrm{P}^{\uparrow} \cup \mathcal{Ran}(p)) \cap (\mathrm{Q}^{\uparrow} \cup \mathcal{Ran}(q))) - \mathcal{Ran}(p \bullet q)} \right.$$

We present $\mathcal{glb}\{\mathbf{P}, \mathbf{Q}\}$ in the alternative notation to clarify the connection with the discussion in Section 4.

Thus, $\langle \mathbb{P}_{\mathcal{U}}, \subseteq \rangle$ is a *lattice*, so we define *addition* of two *problems* as their *lub*, and *intersection* as their *glb; i.e.*: $\mathbf{P} + \mathbf{Q} = \mathcal{lub}\{\mathbf{P}, \mathbf{Q}\}$ and $\mathbf{P} \bullet \mathbf{Q} = \mathcal{glb}\{\mathbf{P}, \mathbf{Q}\}$. Hence,

$\mathbf{R} = \mathbf{P} + \mathbf{Q} \leftrightarrow (D_R = D_P \cup D_Q \wedge R_R = R_P \cup R_Q \wedge r = p + q)$.
$\mathbf{R} = \mathbf{P} \bullet \mathbf{Q} \leftrightarrow (D_R = D_P \cap D_Q \wedge R_R = R_P \cap R_Q \wedge r = p \bullet q)$.

Therefore, $+$ and $\bullet$ are associative, commutative, idempotent, and satisfy the absorption property, i.e., $\mathbf{P} = \mathbf{P} + (\mathbf{P} \bullet \mathbf{Q})$ and $\mathbf{P} = \mathbf{P} \bullet (\mathbf{P} + \mathbf{Q})$.

We can extend the concepts of *addition* and *intersection* from binary to infinitary operations: given $\mathbb{A} \subseteq \mathbb{P}_{\mathcal{U}}$, let $\sum_{\mathbf{P} \in \mathbb{A}} \mathbf{P} = \mathit{lub}\ \mathbb{A}$ and $\bullet_{\mathbf{P} \in \mathbb{A}} \mathbf{P} = \mathit{glb}\ \mathbb{A}$.

Other properties of the *lattice* $\langle \mathbb{P}_{\mathcal{U}}, \subseteq \rangle$ are:

i. it is *complete*,
ii. *problem* $\infty = \langle \mathcal{U}, \mathcal{U}, \mathcal{U} \times \mathcal{U} \rangle$ is its *lub* and *problem* $\mathbf{0} = \langle \emptyset, \emptyset, \mathit{0} \rangle$ is its *glb*,
iii. $\mathbf{P} + \mathbf{0} = \mathbf{P}$ and $\mathbf{P} \bullet \infty = \mathbf{P}$,
iv. it is a *distributive lattice*, i.e., $\mathbf{P} + (\mathbf{Q} \bullet \mathbf{R}) = (\mathbf{P} + \mathbf{Q}) \bullet (\mathbf{P} + \mathbf{R})$, and $\mathbf{P} \bullet (\mathbf{Q} + \mathbf{R}) = (\mathbf{P} \bullet \mathbf{Q}) + (\mathbf{P} \bullet \mathbf{R})$.

5. 2. The Algebraic Structure of the Tarskian Problems of $\mathbb{P}_{\mathcal{U}}$

Let us call $\mathbb{T}_{\mathcal{U}} \subseteq \mathbb{P}_{\mathcal{U}}$ the set of the *T-problems* over $\mathcal{U}$. Clearly, $\langle \mathfrak{R}_{\mathcal{U}}, \subseteq \rangle$ is isomorphic to $\langle \mathbb{T}_{\mathcal{U}}, \subseteq \rangle$. Thus, $\langle \mathfrak{R}_{\mathcal{U}}, \subseteq \rangle$ is a *Complete Boolean Algebra*, but $\langle \mathbb{T}_{\mathcal{U}}, \subseteq \rangle$ is not a *sublattice* of $\langle \mathbb{P}_{\mathcal{U}}, \subseteq \rangle$, as we shall see. The next Theorem is related to the discussion in Section 4, in particular to Remark 4.2 and Lemma 4.3.

Theorem 5.1 The intersection $\mathbf{P} \bullet \mathbf{Q}$ of *T-problems* $\mathbf{P}$ and $\mathbf{Q}$ is a *T-problem* iff $\mathcal{D}om(p) \cap \mathcal{D}om(q) = \mathcal{D}om(p \bullet q)$.

Proof By definition:

$$\mathbf{P} \bullet \mathbf{Q} = p \bullet q \left| \frac{((\mathrm{P}_{\uparrow} \cup \mathcal{D}om(p)) \cap (\mathrm{Q}_{\uparrow} \cup \mathcal{D}om(q))) - \mathcal{D}om(p \bullet q)}{((\mathrm{P}^{\uparrow} \cup \mathcal{R}an(p)) \cap (\mathrm{Q}^{\uparrow} \cup \mathcal{R}an(q))) - \mathcal{R}an(p \bullet q)} \right. \qquad \text{(i)}$$

But, since $\mathbf{P}$ and $\mathbf{Q}$ are *T-problems*, (i) becomes

$$\mathbf{P} \bullet \mathbf{Q} = p \bullet q \left| \frac{(\mathcal{D}om(p) \cap \mathcal{D}om(q)) - \mathcal{D}om(p \bullet q)}{(\mathcal{R}an(p) \cap \mathcal{R}an(q)) - \mathcal{R}an(p \bullet q)} \right. \qquad \text{(ii)}$$

Now, $\mathbf{P} \bullet \mathbf{Q}$ is a *T-problem* iff

$(\mathcal{D}om(p) \cap \mathcal{D}om(q)) - \mathcal{D}om(p \bullet q) = \varnothing$ and
$(\mathcal{R}an(p) \cap \mathcal{R}an(q)) - \mathcal{R}an(p \bullet q) = \varnothing$

Q.E.D.

From this Theorem it is clear that $\mathbb{T}_{\mathcal{U}}$ is not closed under *intersection of problems*. Although $\langle \mathbb{T}_{\mathcal{U}}, \subseteq\rangle$, being isomorphic to $\langle \Re_{\mathcal{U}}, \subseteq\rangle$, is a *complete atomic Boolean algebra*, it is not so with respect to the *subproblem* ordering (i.e., the operations **+** and ●). Furthermore, in view of Theorem 4.4, there is no componentwise manner of determining the *glb* operation induced by the above isomorphism. Notice that the *intersection* **P** ● **Q** of *T-problems* **P** and **Q** is a *T-problem* if *Inj*(**P** ● **Q**).

It should be also noted that the *lub*{**P**, **Q**} = **P** + **Q** of any two *T-problems* is always a *T-problem*. Hence, $\langle \mathbb{T}_{\mathcal{U}}, \subseteq\rangle$ is an *upper subsemilattice* of $\langle \mathbb{P}_{\mathcal{U}}, \subseteq\rangle$.

5.3. Special Problems: Atoms and Zero-like Problems

For every x and y in $\mathcal{U}$, let us denote by $\mathbb{0}_{\bullet}$ the set of the *problems* $\mathbf{0}_x = \langle\{x\}, \varnothing, \mathcal{O}\rangle$, by $\mathbb{0}^{\bullet}$ the set of the *problems* $\mathbf{0}^x = \langle\varnothing, \{x\}, \mathcal{O}\rangle$, and by $\mathbb{0}^{\bullet}_{\bullet}$ the set of the *problems* $\mathbf{0}^y_x = \langle\{x\},\{y\}, \mathcal{O}\rangle$. It is not difficult to see that $\mathbb{0}_{\bullet} \cup \mathbb{0}^{\bullet}$ is the set of *atoms* of $\langle \mathbb{P}_{\mathcal{U}}, \subseteq\rangle$.

Notice that $\mathbb{P}^{\bullet}_{\bullet} = \{\mathbf{P}^y_x : x, y \in \mathcal{U}\}$, with $\mathbf{P}^y_x = \langle\{x\}, \{y\}, \{\langle x, y\rangle\}\rangle$, is the set of *atoms* of the *upper semilattice* $\langle \mathbb{T}_{\mathcal{U}}, \subseteq\rangle$. Nevertheless, the *problems* $\mathbf{P}^y_x \in \mathbb{P}^{\bullet}_{\bullet}$ are not *atoms* of $\langle \mathbb{P}_{\mathcal{U}}, \subseteq\rangle$, since $\mathbf{P}^y_x \bullet \mathbf{P}^z_x = \mathbf{0}_x$, if $y \neq z$, and $\mathbf{P}^y_x \bullet \mathbf{P}^y_x = \mathbf{0}^y$, if $x \neq z$.

Consider now the subset $\mathbb{0}$ of $\mathbb{P}_{\mathcal{U}}$ consisting of the *problems* of the form $\mathbf{0}_D = \langle D, \varnothing, \mathcal{O}\rangle$, $\mathbf{0}^R = \langle\varnothing, R, \mathcal{O}\rangle$ or $\mathbf{0}^R_D = \mathbf{0}_D + \mathbf{0}^R$. It is easy to see that $\langle \mathbb{0}, \subseteq\rangle$ is a *sublattice* of $\langle \mathbb{P}_{\mathcal{U}}, \subseteq\rangle$ whose *glb* is **0**, and whose *lub* is $\mathbf{0}^{\mathcal{U}}_{\mathcal{U}}$.

5.4. Complements and Difference of Problems

Recall the concept of *pseudocomplemented lattice*: $\langle \mathbb{P}_{\mathcal{U}}, \subseteq\rangle$ is *pseudocomplemented* iff for every *problem* **P** there exists $\mathbf{P}^{\mathfrak{s}} = max_{\subseteq}\{\mathbf{Q} \in \mathbb{P}_{\mathcal{U}} : \mathbf{P} \bullet \mathbf{Q} = \mathbf{0}\}$ in $\langle \mathbb{P}_{\mathcal{U}}, \subseteq\rangle$; in such case $\mathbf{P}^{\mathfrak{s}}$ is called a *pseudocomplement* of **P**.

Theorem 5.2. The *problems* of $\langle \mathbb{P}_{\mathcal{U}}, \subseteq\rangle$ that have *pseudocomplements* are exactly the *T-problems*.

Proof Given $\mathbf{P} = \langle D, R, p\rangle$, let $\mathbb{Q} = \{\mathbf{Q} \in \mathbb{P}_{\mathcal{U}} : \mathbf{P} \bullet \mathbf{Q} = \mathbf{0}\}$ and $\mathbf{P}^{\mathfrak{s}} = max_{\subseteq}\mathbb{Q}$, say $\mathbf{P}' = \langle D', R', p'\rangle$.

First, notice that $\mathbb{Q}$ is nonempty, for $\mathbf{0} \in \mathbb{Q}$.

Given an arbitrary triple $\mathbf{Q} = \langle S, T, q \rangle$, we have $\mathbf{Q} \in \mathbb{Q}$ iff the following four conditions hold

(i) $q \subseteq S \times T$

(ii) $D \cap S = \varnothing$

(iii) $R \cap T = \varnothing$

(iv) $p \cap q = \varnothing$.

Given $\mathbf{Q} = \langle S, T, q \rangle \in \mathbb{Q}$, let $\mathbf{Q}_D = \langle \overline{\mathcal{D}om(p)}, T, q \rangle$. Notice that $\mathbf{Q}_D \in \mathbb{Q}$, because, since $p \subseteq D \times R$,

$q \subseteq S \times T \subseteq \overline{D} \times T \subseteq \overline{\mathcal{D}om(p)} \times T$, by (i) and (ii),

$D \cap \overline{\mathcal{D}om(p)} \subseteq D \cap \overline{D} = \varnothing$

$R \cap T = \varnothing$

$p \cap q = 0$.

In view of the maximality of $\mathbf{P}'$, we must have $\mathbf{Q}_D \subseteq \mathbf{P}'$; hence $\overline{\mathcal{D}om(p)} \subseteq D'$.

Since $\mathcal{D}om(p) \subseteq D$, we have $\overline{D} \subseteq \overline{\mathcal{D}om(p)} \subseteq D' \subseteq \overline{D}$.

Hence, $\overline{D} = \overline{\mathcal{D}om(p)}$ and $D = \mathcal{D}om(p)$.

A similar argument with $\mathbf{Q}^R = \langle S, \overline{\mathcal{R}an(p)}, q\rangle$ gives $\overline{\mathcal{R}an(p)} \subseteq R'$, whence $R = \mathcal{R}an(p)$.

This shows that **P** is a *$\mathcal{T}$-problem.*Then, clearly $p' = \bar{p}$

Q.E.D.

Corollary 5.3. The *lattice* $\langle \mathbb{P}_{\mathcal{U}}, \subseteq\rangle$ is not *pseudocomplemented.*

Definition 5.4. Given *problems* **P** and **Q** we define their *difference* **P** – **Q** by

$$\mathbf{P} - \mathbf{Q} = \mathcal{glb}\{\mathbf{S} : \mathbf{S} + (\mathbf{P} \bullet \mathbf{Q}) = \mathbf{P}\} \tag{22}$$

Thus:

$$\mathbf{P} - \mathbf{Q} = p - q\,\left|\frac{(\mathrm{P}_{\uparrow} - \mathrm{Q}_{\uparrow}) - \mathcal{D}om(q)}{(\mathrm{P}^{\uparrow} - \mathrm{Q}^{\uparrow}) - \mathcal{R}an(q)}\right. \tag{23}$$

Definition 5.5. Given *problem* **P**, its *co-pseudocomplement* is the *problem* $\overline{\mathbf{P}} = \infty - \mathbf{P}$.

Thus:

$$\overline{\mathbf{P}} = \bar{p}\,\left|\frac{(\varnothing - \mathrm{P}_{\uparrow}) - \mathcal{D}om(p)}{(\varnothing - \mathrm{P}^{\uparrow}) - \mathcal{R}an(p)}\right. = \bar{p}\,\left|\frac{\varnothing}{\varnothing}\right. \tag{24}$$

i.e., $\overline{\mathbf{P}} = \langle \mathcal{D}om(\bar{p}), \mathcal{R}an(\bar{p}), \bar{p}\rangle$, which is always a *$\mathcal{T}$-problem.* For a *$\mathcal{T}$-problem* **P**, $\overline{\mathbf{P}}$ is its *pseudocomplement* $\mathbf{P}^{\bullet}$.

Notice that Tarski's axioms **At** 2.7 and **At** 2.8 no longer hold on the *lattice* $\langle \mathbb{P}_{\mathcal{U}}, \subseteq\rangle$ and have to be modified. The revised version of **At** 2.8 is $\mathbf{P} \bullet \overline{\mathbf{P}} = \mathbf{0}^{\mathrm{P}^{\uparrow}}_{\mathrm{P}_{\uparrow}}$.The revision of **At** 2.7 amounts to $\mathbf{P} + \overline{\mathbf{P}} = \infty$ (which replaces *complement* by *co-pseudocomplement*). Similarly, Tarski's **At** 2.9 holds for *co-pseudocomplement* : $\bar{\infty} = \mathbf{0}$.

5.5. Complete Subproblems

The notion of *complete subproblem* over $\mathbb{P}_{\mathcal{U}}$ was introduced informally in Section 3. We shall say that **P** is a *complete subproblem* of **Q** iff **P** is a *subproblem* of **Q** and every *Skolem function* of $(\forall d)(d \in D_P \rightarrow (\exists r)(r \in R_P \wedge p(d, r)))$ is total over $\mathcal{D}om(q)$ [Hae87]. Formally:

Definition 5.6 $\mathbf{P} \leftarrow\!\!\bullet\, \mathbf{Q} \leftrightarrow \mathbf{P} \subseteq \mathbf{Q} \wedge \mathcal{D}om(p) = \mathcal{D}om(q)$.

Definition 5.6 states that every *data-result* pair in **P** is in **Q** as well and every *data* connected to some *result* in **Q** is likewise in **P**. If we regard **P** and **Q** as *models* of *specifications*, then **Q** is weaker (more general) than **P**.

Consider now the set $\mathbf{P}_{\leftarrow\bullet}$ of all *complete subproblems* of a given *problem* **P**. The *glb* of any two *complete subproblems* **Q** and **R** of **P** is a *complete subproblem* of **P** iff $\mathcal{D}om(q \cap r) = \mathcal{D}om(q) = \mathcal{D}om(r)$. So, $\mathbf{P}_{\leftarrow\bullet}$ is not closed under *intersection*. On the other hand, the *lub* of any two *complete subproblems* **Q** and **R** of **P** is always a *complete subproblem* of **P**. Thus, for any *problem* **P** , $\langle \mathbf{P}_{\leftarrow\bullet}, \leftarrow\bullet \rangle$ is an *upper semilattice.*

Denoting by $\mathbf{P}_+$ the set of *subproblems* of **P**, $\langle \mathbf{P}_+, \subseteq \rangle$ is *a complete lattice*. Moreover, $\langle \mathbf{P}_{\leftarrow\bullet}, \leftarrow\bullet \rangle$ is an *upper subsemilattice* of $\langle \mathbf{P}_+, \subseteq \rangle$.

Notice that $\mathbf{P}_{\leftarrow\bullet}$ and $\mathbf{P}_+$ are not closed under *difference*.

5.6 Converse and Relative Product

The *converse* of the *problem* $\mathbf{P} = \langle D_P, R_P, p \rangle$ is the problem $\tilde{\mathbf{P}} = \langle R_P, D_P, \tilde{p} \rangle$.

Definition 5.7 The *relative product* of *problems* **P** and **Q** is the *problem*

$$\mathbf{P} \,;\, \mathbf{Q} = \langle D_P, R_Q, p \,;\, q \rangle$$

Thus:

$$\mathbf{P} \,;\, \mathbf{Q} \;=\; p \,;\, q \left| \frac{(\mathrm{P}_{\uparrow} \cup \mathcal{D}om(p)) - \mathcal{D}om(p \,;\, q)}{(\mathrm{Q}^{\uparrow} \cup \mathcal{R}an(q)) - \mathcal{R}an(p \,;\, q)} \right. \tag{25}$$

It is interesting to observe that Tarski's axioms **At** 2.11 and 2.12, as well as Theorems 2.6, 2.8, and 2.9 hold in our theory, while the equivalent of **At** 2.13 should state $\mathbf{P} \,;\, \mathbf{1}^{\mathrm{P}} = \mathbf{P}$ (where $\mathbf{1}^{\mathrm{P}}$ is another way of denoting $\mathbf{1}_{R_P}$, i.e., the identity on R_P; we will also use $\mathbf{1}_P$ instead of $\mathbf{1}_{D_P}$). Axiom **At** 2.14 holds for simple relation algebras [Sch85], which in general is not the case with program derivation, since our base set amounts to the union of the various sorts of a given many-sorted algebra (cf. the reduction of many-sorted logic to one-sorted logic [End72]. Of course, one way of fixing it is by relativizing it to appropriate domains and ranges.

The equivalents, in our theory, of Tarski's Theorems 2.7, 2.10, 2.12, and 2.13, are:

$$\mathbf{P} \,;\, \mathbf{0} = \mathbf{0}_{D_P}$$
$$\mathbf{0} \,;\, \mathbf{P} = \mathbf{0}^{R_P}$$
$$\mathbf{1}_P \,;\, \mathbf{P} = \mathbf{P}$$
$$(\infty \,;\, \mathbf{P}) \bullet \mathbf{Q} \in \mathbb{O} \rightarrow (\infty \,;\, \mathbf{Q}) \bullet \mathbf{P} \in \mathbb{O}$$

Finally, we denote by $\mathbf{P}^{;n}$ the product of **P** by itself n times, i.e. $\mathbf{P}^{;0} = \mathbf{1}^{\mathrm{P}}$ and $\mathbf{P}^{;n+1} = \mathbf{P} \,;\, \mathbf{P}^{;n}$, and by $\mathbf{P}^{;*}$ the *closure* $\sum_{n \in \mathrm{N}} \mathbf{P}^{;n}$.

5.7 Direct Product, Fork and Projections

The *direct product* of *problems* is analogous to the equivalent operation on *relations*, introduced informally in Section 3. The formal definition is as follows.

Definition 5.8 We define the *direct product* of *problems* $\mathbf{P}$ and $\mathbf{Q}$ as the *problem* $\mathbf{P} \times \mathbf{Q}$ whose components are $D_{P \times Q} = D_P \times D_Q$, $R_{P \times Q} = R_P \times R_Q$ and its *condition* $q_{P \times Q} = p \times q$ (where $p \times q$ is the operation on *relations* defined in Section 3).

The *direct product* distributes over sum on both sides, i.e.,
$\mathbf{P} \times (\mathbf{Q} + \mathbf{R}) = \mathbf{P} \times \mathbf{Q} + \mathbf{P} \times \mathbf{R}$ and $(\mathbf{Q} + \mathbf{R}) \times \mathbf{P} = \mathbf{Q} \times \mathbf{P} + \mathbf{R} \times \mathbf{P}$.

Moreover, $\mathbf{P} \times \mathbf{0} = \mathbf{0} \times \mathbf{P} = \mathbf{0}$, and $\mathbf{P} \times \mathbf{0}_{D_P, R_P} = \mathbf{0}_{D_P, R_P} \times \mathbf{P}$ both being the *problem* $\mathbf{0}_{D_P \times D_P\ ,\ R_P \times R_P}$.

Finally, a kind of distributivity property over ; is $(\mathbf{P}\ ;\ \mathbf{Q}) \times (\mathbf{R}\ ;\ \mathbf{S}) = (\mathbf{P} \times \mathbf{R})\ ;\ (\mathbf{Q} \times \mathbf{S})$. This is an example of a property that hinges on the non-associativity of the universe structuring operation $*$ (with associative $*$ we would have just an inclusion).

As in the case of *relations*, a normal form for $\mathbf{P} \times \mathbf{Q} \times \mathbf{R}$ is $\mathbf{P} \times (\mathbf{Q} \times \mathbf{R})$. So, $\mathbf{P}^{\times n}$ will be a notation for the *Direct Product* of $\mathbf{P}$ by itself n times. Finally, we define $\mathbf{P}^{\times *} = \sum_{n \in N} \mathbf{P}^{\times n}$.

Definition 5.9 We define the *fork* of problems $\mathbf{P}$ and $\mathbf{Q}$ as the problem $\mathbf{P} \nabla \mathbf{Q}$ the components of which are $D_{P \nabla Q} = D_P \cap D_Q$, $R_{P \nabla Q} = R_P \times R_Q$ and $q_{P \nabla Q} = p \nabla q$ (where $p \nabla q$ is the operation on *relations* defined in Section 3).

As in the case of *relations*, for the special case of $\mathbf{P} = \mathbf{Q} = \mathbf{1}$, we introduce $\mathbf{2} = \mathbf{1} \nabla \mathbf{1}$, $\mathbf{3} = \mathbf{1} \nabla \mathbf{2}$, $\mathbf{4} = \mathbf{1} \nabla \mathbf{3}$, etc. So, $\mathbf{3} = \langle \mathcal{U}, \mathcal{U} \times (\mathcal{U} \times \mathcal{U}), \{\langle u, (u * (u * u))\rangle\}\rangle$.

By the *first-projection* Π_1 we mean the *problem* with *data carrier* $\mathcal{U} \times \mathcal{U}$, *result carrier* $\mathcal{U}$, and whose condition is the *first-projection* relation introduced in Section 3. Similarly, we define $\Pi_2 = \widetilde{\infty} \nabla \mathbf{1}$

We should notice that the distributive properties introduced in Section 3 for *forks, direct products* and *projections* over *relations* will also hold when appropriately expressed for *problems*.

5.8 Why Problems?

We shall now give some justification for the use of problems, by reviewing the discussion in Section 4 motivating their introduction, and indicating some reasons for defining the operations on them as done in the preceding parts of this Section.

Let us start from the discussion in Section 4. One can say that it forced the evolution of the idea of the behavior of a program (and of a specification, as well), from a simple binary relation p, via a preconditioned relation (a pair $\langle \mathcal{W}, p \rangle$), to a partial relation, i. e. a problem $\mathbf{P} = \langle \mathrm{D}_\mathbf{P}, \mathrm{R}_\mathbf{P}, p \rangle$. In this sense we have gone from Tarski's binary relations to *partial binary relations.*

But we can always go back to the original binary relations. If we call problems **P** and **Q** relationally equivalent iff they have the same condition, it is clear that this partitions $\mathbb{P}_\mathcal{U}$ into equivalence classes, each one of them consisting of the problems with the same condition. Thus, the quotient structure will be in one-to-one correspondence with $\Re_\mathcal{U}$. Now, we can express relational equivalence within our calculus, because $q \subseteq p$ if and only if $\mathbf{Q} + 0^{\mathrm{R}_\mathbf{P}}_{\mathrm{D}_\mathbf{P}} \subseteq \mathbf{P} + 0^{\mathrm{R}_\mathbf{Q}}_{\mathrm{D}_\mathbf{Q}}$, i. e. $\mathbf{Q} + 0^{\mathcal{U}}_{\mathcal{U}} \subseteq \mathbf{P} + 0^{\mathcal{U}}_{\mathcal{U}}$.

Now, recall that $\mathbf{P}_+$ is the sublattice of the subproblems of **P** and $\mathbb{T}_\mathcal{U}$ is the *upper semilattice* of the Tarskian problems, so that $\mathbf{P}_+ \cap \mathbb{T}_\mathcal{U}$ is the *upper semilattice* of the Tarskian subproblems of **P**. Let $\mathring{\mathbf{P}}$ be the largest T-subproblem of **P**. This provides us with a particularly simple algebraic description for the class of problems relationally equivalent to **P**, namely $\{\mathring{\mathbf{P}} + 0^{\mathrm{R}}_{\mathrm{D}} : 0^{\mathrm{R}}_{\mathrm{D}} \in \mathbb{O}\}$. This description clarifies why the effect of identifying relationally equivalent problems leads us to the Tarskian problems, for then $\mathbb{O}$ becomes the class of problems with condition *0.*

It is now easy to see how we can express with our problem-theoretic notation, some basic concepts of correctness. Consider a specification with behavior **P** and a program with behavior **Q** as well as a set $\mathcal{W}$. Then, we can express

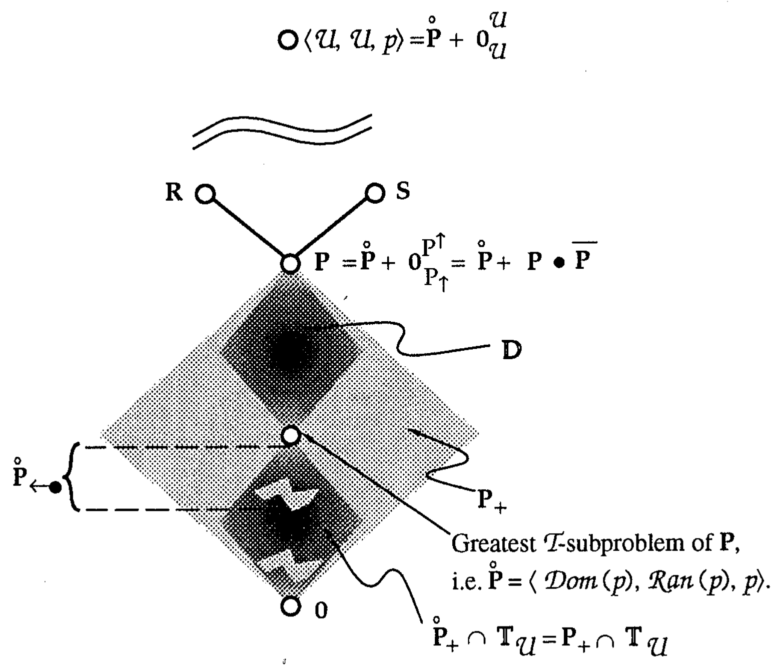
- partial correctness of the program with respect the specification by $\mathbf{1}_{\mathrm{D}_\mathbf{P}} ; \mathbf{Q} \subseteq \mathbf{P} + 0^{\mathrm{R}_\mathbf{Q}}$;
- termination of the program over $\mathcal{W}$ by $\mathbf{1}_\mathcal{W} \subseteq \mathbf{P} ; \tilde{\mathbf{P}}$;
- total correctness of the program with respect the specification by $\mathbf{1}_{\mathrm{D}_\mathbf{P}} ; \mathbf{Q} \leftarrow\!\bullet\ \mathbf{P} + 0^{\mathrm{R}_\mathbf{Q}}$.

Now, consider specifications, denoting problems **R** and **S**. As in our example in Section 4., let $\mathbf{P} = \mathbf{R} \bullet \mathbf{S} = \langle \mathrm{D}_\mathbf{P}, \mathrm{R}_\mathbf{P}, p \rangle$. Our considerations can be summarized in Figure 5.1. The ideal program, fitting **P** like a glove, would have behavior $\mathring{\mathbf{P}} = \langle \mathcal{D}om(p), \mathcal{R}an(p), p \rangle$. Notice that the Tarskian subproblems of $\mathring{\mathbf{P}}$ form the upper subsemilattice $\mathring{\mathbf{P}}_+ \cap \mathbb{T}_\mathcal{U} = \mathbf{P}_+ \cap \mathbb{T}_\mathcal{U}$. Also, if a program has its behavior

- in $\mathbf{P}_+$ then it is partially correct with respect to **P** and has *data indefinition set* upper-bounded by $\mathrm{D}_\mathbf{P} = \mathrm{D}_\mathbf{R} \cap \mathrm{D}_\mathbf{S}$;

- in $\mathbf{P}_+ \cap \mathbb{T}_{\mathcal{U}}$ then it is partially correct with respect to **P** and terminates over its precondition;
- in $\mathring{\mathbf{P}}_{\leftarrow\bullet}$ then it is totally correct with respect to **P**;
- in $\mathbb{D}$ then it is partially correct with respect to **P** and is of the form $\mathring{\mathbf{P}} + 0^{\mathrm{R}}_{\mathrm{D}} : 0^{\mathrm{R}}_{\mathrm{D}} \in 0^{\mathrm{R}}_{\mathrm{D}+}$;

Also, $0\langle \mathcal{U}, \mathcal{U}\ p\rangle = \mathring{\mathbf{P}} + 0^{\mathcal{U}}_{\mathcal{U}}$ is the lub of the behaviors of the programs partially correct with respect to **P**.



$\mathrm{O}\langle \mathcal{U}, \mathcal{U}, p\rangle = \mathring{\mathbf{P}} + 0^{\mathcal{U}}_{\mathcal{U}}$
R
S
$\mathbf{P} = \mathring{\mathbf{P}} + 0^{\mathrm{P}\uparrow}_{\mathrm{P}\uparrow} = \mathring{\mathbf{P}} + \mathbf{P} \bullet \overline{\mathbf{P}}$
D
$\mathring{\mathbf{P}}_{\leftarrow\bullet}$
$\mathbf{P}_+$
Greatest $\mathcal{T}$-subproblem of **P**,
i.e. $\mathring{\mathbf{P}} = \langle \mathcal{D}om\,(p), \mathcal{R}an\,(p), p\rangle$.
0
$\mathring{\mathbf{P}}_+ \cap \mathbb{T}_{\mathcal{U}} = \mathbf{P}_+ \cap \mathbb{T}_{\mathcal{U}}$

Figure 5.1

Finally, notice that we can give problem-theoretic characterization for several concepts introduced in this section. For instance.

P is viable iff $\mathbf{1_P} \subseteq \mathbf{P}\,;\tilde{\mathbf{P}}$.

P is deterministic iff $\mathbf{P} = \mathbf{P}\,;\tilde{\mathbf{P}}\,;\,\mathbf{P}$.

$\mathbf{P} \subseteq \mathbf{Q}$ iff $\mathbf{P} + \mathbf{Q} = \mathbf{Q}$.

$\mathbf{P} \leftarrow\!\bullet\ \mathbf{Q}$ iff $\mathbf{P} \subseteq \mathbf{Q}$ and $\mathbf{Q}\,;\infty \subseteq \mathbf{P}\,;\infty$.

6. EXPRESSIVENESS

In this Section we shall examine the expressiveness of our *Language of Problems.* The importance of this question stems from the following fact. A *problem* **P** is a triple

$\mathbf{P} = \langle D_P, R_P, p \rangle$, where the *carriers* D_P and R_P are sets and the *condition* $p \subseteq D_P \times R_P$ is a binary relation. One often specifies the *problem condition* by means of a logical formula describing how input data are connected to output results. During the derivation process we would like to manipulate the *problem* as an single entity by means of our operations. The question is then whether we can do this all along the process, or whether we will have, from time to time, to examine its components, especially the formula describing its *condition*. The latter would be undesirable, since it amounts to shifting between levels. This undesirable detour would be avoided if we could express our *problem*, from the very start, entirely within our language, in terms of some given basic *problems*. So, the question we wish to address here is : "can we express any *condition* defined by a first-order formula in our *problem-theoretic* language ?".

At first sight, the answer would appear to be negative. The reason for this suspicion is as follows. Tarski has examined the question of the expressiveness of his *relational calculus* [Tar41]. He asked whether every property of *relations*, relation among *relations*, etc., that can be defined in his *Elementary Theory of Relations*, can be expressed in his *Calculus of Relations*. Tarski's answer to this question is no. According to him, even simple expressions like :

$$(\forall x)(\forall y)(\forall z)(\exists u)(r(x, u) \wedge r(y, u) \wedge r(z, u)) \tag{26}$$

$$(\exists x)(\exists y)(\exists z)(\exists u)(r(x, y) \wedge r(x, z) \wedge r(x, u) \wedge r(y, z) \wedge r(y, u) \wedge r(z, u)) \tag{27}$$

cannot be expressed within his *relational calculus*. For instance, no sentence of his *Calculus of Relations* is satisfied by exactly the same *relations* that satisfy expression (26).

Let us see how we can express (26) within our language. In fact, we will show two alternative ways to express (26).

One way to proceed is as follows. First, notice that (26) is equivalent to

$$(\forall x)(\forall y)(\forall z)(\exists u)(\exists v)(\exists w)(r(x, u) \wedge r(y, v) \wedge r(z, w) \wedge u = v \wedge u = w)$$

Now, $u = v \wedge u = w$ is equivalent to $u \, \mathit{3} \, u{*}(v{*}w)$, i.e., $u{*}(v{*}w) \tilde{\mathit{3}} u$ (recall that we are using either $r(x, y)$ or $x \; r \; y$ to express the fact $\langle x, y \rangle \in r$).

Thus, $r(x, u) \wedge r(y, v) \wedge r(z, w) \wedge u = v \wedge u = w$ is equivalent to

$$x{*}(y{*}z) \; (r \times r \times r) \; u{*}(v{*}w) \wedge u{*}(v{*}w) \, \tilde{\mathit{3}} \, u$$

Therefore, (26) is equivalent to the totality of the *relation* $q = (r \times r \times r) \,;\, \tilde{\mathit{3}}$, which can be expressed by $\mathit{1} \times \mathit{1} \times \mathit{1} \subseteq q \,;\, \tilde{q}$.

For another way of expressing (26), we start by noticing that $r(x, u) \wedge r(y, u) \wedge r(z, u)$ is equivalent to $x\ r\ u \wedge u\ \tilde{r}\ y \wedge u\ \tilde{r}\ z$. Thus, (26) amounts to $(\forall x)(\forall y)(\forall z)(x\ p\ y{*}z)$, where p is the *relation* $r;\, 2\,;\, (\tilde{r} \times \tilde{r})$. So, all we have to say is that *relation* p is universal over the appropriate *data* and *result carriers*, namely, $p = \infty \nabla \infty$, i. e., $p = 2\,;\, (\infty \times \infty)$.

A similar reasoning could be applied to (27) to provide an expression for it within our problem-theoretic formalism. It may be instructive to examine why we have succeeded in expressing (26) within our formalism and how we have overcome the difficulties encountered in trying to express it in Tarski's relational calculus. First, notice that there is no difficulty in expressing a simpler version of (26), like

$$(\forall x)(\forall y)(\exists u)(r(x, u) \wedge r(y, u))$$

Since this is equivalent to

$$(\forall x)(\forall y)(\exists u)(r(x, u) \wedge \tilde{r}(u, y))$$

it can be expressed by $r;\, \tilde{r} = \infty$. The key idea here is the fact that the existential quantifier $(\exists u)$ can be simulated by the *relative* product. If we try to apply this simple idea to (26), we would be led to something like

$$(\forall x)(\forall y)(\forall z)(\exists u)(r(x, u) \wedge \tilde{r}(u, y) \wedge \tilde{r}(u, z))$$

This is equivalent to (26), but we cannot simulate the effect of $(\exists u)$ by the relative product. The reason for this is the fact that variable u now occurs three times in the matrix of the formula. Tarski's relational calculus has no variables over individuals, and the relative product $r\, ;\, \tilde{r}$ "consumes", so to speak, variable u. We circumvent this difficulty by making copies of it, by means of the special relation *2*, and then using the two copies distinctly, which we can do by means of the direct product of relations. Finally, the *1*'s are handy to express that the relation obtained is universal over a certain domain. Thus, we see that the addition of extra operations and constants strictly increases the expressiveness of the language of relations.

Encouraged by these positive results, one is tempted to conjecture that any *problem* that can be described within first-order logic has an equivalent *problem-theoretic* formulation. We will show that indeed this is so. This will be done by means of two reductions: the first one will show that we do not have to worry about the structure of inputs and outputs, and the second one will enable us to handle inputs and outputs in the same manner.

First, we should clarify what we mean by a *problem* described within a first-order language $\mathcal{L}$. The central component in a problem is its *condition*, which can be naturally defined by a first-order formula φ of $\mathcal{L}$, describing how input and output values are to be related. But, one also has to describe the data and result carriers; this can be done by giving the structure of their elements in terms of their components. These ideas are embodied in the concept of problem presentation. A *problem presentation* is a triple P describing the three components of a *problem* as follows

input : $s(x_1, \ldots, x_k)$, with $x_i : W_i$;
output : $t(y_1, \ldots, y_l)$, with $y_j : W'_j$;
condition : $\varphi(x_1, \ldots, x_k, y_1, \ldots, y_l)$;

where $s(x_1, \ldots, x_k)$ and $t(y_1, \ldots, y_l)$ are terms constructed from these variables by means of the universe structuring operation $*$, W_i and W'_j are basic sorts, and the condition is given by a formula of $\mathcal{L}$.

Of course, the idea is that such a *presentation* describes a *problem* on a structure for the language. Given a structure $\mathfrak{B}$ for $\mathcal{L}$ with domain B, we have $W_i^{\mathfrak{B}} \subseteq B$ and $W'_j{}^{\mathfrak{B}} \subseteq B$. The *problem defined* by the *presentation* P on $\mathfrak{B}$ is the *problem* $P^{\mathfrak{B}} = \langle D_P^{\mathfrak{B}}, R_P^{\mathfrak{B}}, p^{\mathfrak{B}} \rangle$, where the data carrier $D_P^{\mathfrak{B}} = \{s^{\mathfrak{B}}(d_1, \ldots, d_k) : x_i \in W_i^{\mathfrak{B}}\}$, and similarly for the result carrier $R_P^{\mathfrak{B}}$; and its condition $p^{\mathfrak{B}}$ consists of the input-output pairs such that $\mathfrak{B} \models \varphi[d_1, \ldots, d_k, r_1, \ldots, r_l]$. Here, $\mathfrak{B} \models \varphi[b_1, \ldots, b_m]$ means that structure $\mathfrak{B}$ satisfies formula φ under the assignment of $b_1, \ldots, b_m$, respectively, to the variables $v_1, \ldots, v_m$ [Ebb80, End72]. Thus, up to internal structuring, $p^{\mathfrak{B}}$ can be regarded as ${}_k\varphi_l{}^{\mathfrak{B}} = \{\langle (d_1, \ldots, d_k), (r_1, \ldots, r_l)\rangle : (d_1, \ldots, d_k, r_1, \ldots, r_l) \in \varphi^{\mathfrak{B}}\}$, where $\varphi^{\mathfrak{B}}$ is the (k+l)-ary relation defined by formula φ on structure $\mathfrak{B}$.

We are now ready for our fist reduction. As hinted at above, it amounts to putting the input and output into "normal forms", thereby getting rid of the above terms s and t.

Lemma 6.1 Given *problem presentations* P with

input : $s(x_1, \ldots, x_k)$, with $x_i : W_i$;
output : $t(y_1, \ldots, y_l)$, with $y_j : W'_j$;
condition : $\varphi(x_1, \ldots, x_k, y_1, \ldots, y_l)$;

and N_φ with

input : $x_1 * (\ldots (x_{k-1} * x_k) \ldots)$;
output : $y_1 * (\ldots (y_{l-1} * y_l) \ldots)$;
condition : $\varphi(x_1, \ldots, x_k, y_1, \ldots, y_l)$;

there exist problem-theoretic terms $\mathbf{1}_{\mathcal{D}}$, S, T and $\mathbf{1}_{\mathcal{R}}$, such that
$P = \mathbf{1}_{\mathcal{D}} ; \widetilde{S} ; N_\varphi ; T ; \mathbf{1}_{\mathcal{R}}$.

Proof idea Terms $\mathbf{1}_{\mathcal{D}}$, S, T and $\mathbf{1}_{\mathcal{R}}$ are easily constructed by means of projections, forks and the identities on the basic sorts, by taking into account that, for instance, T is to convert an object of the form $y_1 * (\ldots (y_{l-1} * y_l) \ldots)$ into $t(y_1, \ldots, y_l)$.
Q.E.D.

Our second reduction will show that any such "normalized" problem can be reduced (by reducing ${}_k\varphi_l$ to φ) to a special kind of problem, which essentially checks whether a data-result combination is acceptable, and is thus called a characteristic problem. By a *characteristic problem* we mean a problem with $D_P = R_P = \mathcal{U}$ whose condition is a filter (i. e., an identity relation) over a subset of $\mathcal{U}$, where, as before, $\mathcal{U} = B^*$ is the *free groupoid* generated by the base set B.

Lemma 6.2 Given a *problem presentations* N_φ with

input : $x_1 * (\ldots (x_{k-1} * x_k) \ldots);$
output : $y_1 * (\ldots (y_{l-1} * y_l) \ldots);$
condition: $\varphi(x_1, \ldots, x_k, y_1, \ldots, y_l);$

and χ_φ with

input: $x_1 * (\ldots (x_{k-1} * (x_k * (y_1 * (\ldots (y_{l-1} * y_l) \ldots)) \ldots);$
output: $x_1 * (\ldots (x_{k-1} * (x_k * (y_1 * (\ldots (y_{l-1} * y_l) \ldots)) \ldots);$
condition: $\varphi(x_1, \ldots, x_k , y_1, \ldots, y_l);$

there exist problem-theoretic terms A and Ω such that $N_\varphi = \widetilde{A}\ ;\ \chi_\varphi\ ;\ \Omega$.

Proof idea Terms A and Ω are easily constructed by means of projections and forks and the identities on the basic sorts, by taking into account that, for instance, Ω is to convert $x_1 * (\ldots (x_{k-1} * (x_k * (y_1 * (\ldots (y_{l-1} * y_l) \ldots)) \ldots) * y_1 * (\ldots (y_{l-1} * y_l) \ldots)$ into $t(y_1 * (\ldots (y_{l-1} * y_l) \ldots)$.
Q.E.D.

We are now ready for our main expressiveness result. In view of the above Lemmas, it suffices to consider characteristic problems. The next Theorem will show that every characteristic problem whose condition is expressed by a first-order formula φ can be denoted by a term $\Xi\ (\varphi)$ on problems, provided that we have a constant for each basic predicate symbol.

$$\begin{tikzcd}
\text{formula} & & \text{term} \\
\varphi \arrow[rr, "\Xi"] \arrow[d, "\mathfrak{B}"] & & \Xi(\varphi) \arrow[d, "\mathfrak{B}"] \\
\chi_\varphi{}^{\mathfrak{B}} \arrow[rr, phantom, "="] & & \mathfrak{B}\,[\Xi_\varphi] \\
\text{characteristic problem} & & \text{problem denoted}
\end{tikzcd}$$

Theorem 6.3 Given any *characteristic problem* **P** whose *condition* is expressed by a formula φ of a first-order language $\mathcal{L}$, if for every basic predicate occurring in φ we have a constant on *problems*, then **P** can be expressed by a term on *problems*.

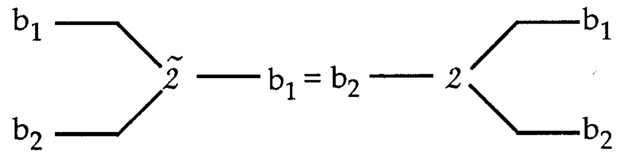
Proof outline

We will construct, by induction on the structure of formula φ, a problem-theoretic term $\Xi\,(\varphi)$ such $\mathfrak{B}[\Xi\,(\varphi)] = \chi_\varphi{}^{\mathfrak{B}}$.

For the sake of clarity, we shall indicate the main argument line, without worrying too much about the variables. These details amount to an exercise in dealing with projections and permutations.

Basis (atomic formulas): Let us distinguish two cases.

Case φ is $x = y$: $\Xi\,(x = y) := \tilde{2}\,;\,\mathbf{2}$



Case φ is $\rho(v_1, \ldots, v_n)$.

By assumption we have a constant problem $\Xi(\rho)$ for each basic predicate ρ.

Inductive step

Case φ is $\neg\,\theta$: $\quad \Xi\,(\varphi) := \overline{\Xi\,(\theta)} \bullet \mathbf{1}$

Case φ is $\psi \vee \theta$: $\quad \Xi\,(\psi \vee \theta) := \Xi\,(\psi) + \Xi\,(\theta)$

Case φ is $\exists y\ \theta(x, y)$: $\quad \Xi(\exists y\ \theta(x, y)) := \tilde{\Pi}_1\,;\,\Xi\,(\theta)\,;\,\Pi_1$

$$\begin{tikzcd}[arrows=-]
 & & b_1 \arrow[r] & b_1 \arrow[dr] & & \\
b_1 \arrow[r] & \tilde{\Pi}_1 \arrow[ur] \arrow[dr] & \boxed{\Xi_\theta} & & \Pi_2 \arrow[r] & b_1 \\
 & & b_2 \arrow[r] & b_2 \arrow[ur] & &
\end{tikzcd}$$

Notice that the above expression is of the form *generate and test,* where *generate* is $\tilde{\Pi}_1 = \mathbf{1} \nabla \infty$ and *test* is Ξ_θ, Π_1 being the *extraction function.* The general case follows the above idea. The rearrangement of variables can be expressed by a suitable combination of projections and direct products (*formal noise* again).

Q.E.D.

The above Theorem shows that we can express the effect of connectives and quantifiers within our problem-theoretic language. Similarly to the cases of disjunction , we have:

$$\Xi(\psi \wedge \theta, n) = \Xi(\psi, n) \bullet \Xi(\theta, n)$$

Much as the relative product is central in expressing the effect of existential quantification, the relative sum is used for the universal quantifier.

Notice that our Theorem 6.3 suggests that we have a new candidate for an algebra of first-order logic, akin to cylindric and polyadic algebras. It presents, however, the advantage of being a finite extension of Tarski's algebras of binary relations. Further details can be found in [Vel90].

By putting together the two Lemmas and the preceding Theorem, we immediate conclude that the language of *problem presentations* can be interpreted into our language of problem theoretic terms.

Theorem 6.4 Let $\mathcal{L}$ be a first-order language and assume that for each basic predicate ρ of $\mathcal{L}$ we have a term for the corresponding characteristic problem χ_ρ. Then, for every problem presentation P there exist a problem-theoretic term T(P) denoting the same problem, in the sense that for every structure $\mathfrak{B}$ for $\mathcal{L}$: $P^{\mathfrak{B}} = \mathfrak{B}\,(T(P))$.

In fact, our argument indicates the construction of the interpretation T.

As a simple example, $\tilde{\mathbf{2}}\,;\mathbf{2}$ is an equality filter over the entire universe $\mathcal{U}$. If one needs an equality filter for, say, lists, as in the p*alindrome problem,* it suffices to restrict it to the sort of lists, thereby obtaining $(\mathbf{1}_{\mathcal{L}^*} \times \mathbf{1}_{\mathcal{L}^*})\,;\tilde{\mathbf{2}}\,;\,\mathbf{2}\,;\,\mathbf{1}_{\mathcal{L}^*}$.

7. MONOTONICITY AND CONTINUITY

Operations $+, \bullet, \times, \nabla, \widetilde{\ }$ and $;$ are *monotonic* and *continuous* with respect to the ordering $\subseteq$ over $\mathbb{P}\mathcal{U}$. Hence we call these operations algorithmic.

For any $\mathbf{P} \in \mathbb{P}\mathcal{U}$, + and $\times$ are *monotonic*, and ; is *right monotonic* over $\langle \mathbf{P}_{\leftarrow\!\bullet}, \leftarrow\!\bullet \rangle$. As one might suspect, the *left monotonicity* of ; over $\langle \mathbf{P}_{\leftarrow\!\bullet}, \leftarrow\!\bullet \rangle$ cannot be taken for granted. Given any three *problems* **P**, **Q**, and **R**, with *conditions* p, q and r, respectively; then:

$$\text{If } \mathbf{P} \leftarrow\!\bullet\ \mathbf{Q} \text{ then } \mathbf{P} ; \mathbf{R} \leftarrow\!\bullet\ \mathbf{Q} ; \mathbf{R} \text{ iff } \mathcal{D}om(q ; r) \subseteq \mathcal{D}om(p ; r) \tag{28}$$

Evidently, in view of the result of Theorem 6.3 above, we should express this condition entirely within the *Algebraic Theory of Problems*. Indeed, we can write:

$$\text{If } \mathbf{P} \leftarrow\!\bullet\ \mathbf{Q} \text{ then } \mathbf{P} ; \mathbf{R} \leftarrow\!\bullet\ \mathbf{Q} ; \mathbf{R} \text{ iff}$$
$$(\mathrm{Q} ; \mathrm{R} ; (\widetilde{\mathrm{Q} ; \mathrm{R}})) \bullet \mathbf{1} \subseteq (\mathrm{P} ; \mathrm{R} ; (\widetilde{\mathrm{P} ; \mathrm{R}})) \bullet \mathbf{1}$$

which, by applying axiom **At** 2.11, turn to be:

$$\text{If } \mathrm{P} \leftarrow\!\bullet\ \mathrm{Q} \text{ then } \mathbf{P} ; \mathbf{R} \leftarrow\!\bullet\ \mathbf{Q} ; \mathbf{R} \text{ iff}$$
$$(\mathrm{Q} ; \mathrm{R} ; \widetilde{\mathrm{R}} ; \widetilde{\mathrm{Q}}) \bullet 1 \subseteq (\mathrm{P} ; \mathrm{R} ; \widetilde{\mathrm{R}} ; \widetilde{\mathrm{P}}) \bullet 1$$

A somewhat more compact way of expressing (28) may be:

$$\text{If } \mathrm{P} \leftarrow\!\bullet\ \mathrm{Q} \text{ then } \mathrm{P} ; \mathrm{R} \leftarrow\!\bullet\ \mathrm{Q} ; \mathrm{R} \text{ iff } \mathrm{Q} ; \mathrm{R} ; \infty \subseteq \mathrm{P} ; \mathrm{R} ; \infty$$

These expressions for the condition $\mathcal{D}om(q ; r) \subseteq \mathcal{D}om(p ; r)$ are equivalent, because

$$(\mathrm{Q} ; \mathrm{R} ; \widetilde{\mathrm{R}} ; \widetilde{\mathrm{Q}}) \bullet 1 \subseteq (\mathrm{P} ; \mathrm{R} ; \widetilde{\mathrm{R}} ; \widetilde{\mathrm{P}}) \bullet 1,$$
$$\mathrm{Q} ; \mathrm{R} ; \infty \subseteq \mathrm{P} ; \mathrm{R} ; \infty.$$

Conditions for the monotonicity of the other algorithmic operations can be obtained in a similar manner.

8. SPECIFICATIONS, PROGRAMS AND DERIVATION

Let us recapitulate the main lines of our development so far. We have indicated, in Section 3, how Tarski's calculus of binary relations can be extended by new operations on a structured universe so as to become appropriate for program derivation. Then, we have shown, in Section

4, that the association of a precondition to a binary relation forces the calculus to deal with partial relations. Such partial relations are basically what we call problems, whose theory has been partly presented in Section 5. Section 6 has reassured us that we do not have to worry about the expressiveness of our problem-theoretic language: it is at least as expressive as first-order logic. On the other hand, in Section 7, we have examined monotonicity and continuity of our operations on problems, which will be useful now in guaranteeing that recursive expressions have fixpoints, thereby ensuring that programs written in our language will have computational meaning.

In this Section we shall examine some aspects of using our problem-theoretic language and calculus for specifying and constructing programs. We start with some considerations on specification and programming languages. Then, we examine some issues concerning the use of our problem-theoretic language as a specification language and as a programming language, as well the use of our problem-theoretic calculus for program derivation. We should, however, stress that our problem-theoretic concepts provide not "yet another specification / programming language or derivation calculus". What they do provide is rather a framework for expressing, and reasoning about, the semantics of specification and programming languages. In this sense, our considerations will be perfectly general.

8.1. Programming and Specification Languages

The first question that arises is "Why does one derive?" A first answer to this question might be simply "Because the specification and programming languages are different". But, consider again our *palindrome problem* in Section 3. Why do we not use as a program the very term (1) defining our problem at the beginning of the derivation process? An answer is that we must eliminate from such a term those operations on *problems* that are not algorithmic. For instance, if we recall the definition $\mathbf{P} - \mathbf{Q} = \mathit{glb}\{\mathbf{S} : \mathbf{S} + (\mathbf{P} \bullet \mathbf{Q}) = \mathbf{P}\}$, it should be clear that difference does not deserve to be called algorithmic. But, the first term in the derivation of the *palindrome problem*, namely, $\mathbf{Pal} = \mathbf{2} \,;\, (\mathbf{1} \times \mathbf{Rev}) \,;\, \mathbf{Eql}$ does not contain any non-algorithmic operation. In this case there is another reason for deriving.

Let us compare the first and last expressions of the derivation of Section 3, written now as terms over *problems*, namely

$$\mathbf{Pal} = \mathbf{2} \,;\, (\mathbf{1} \times \mathbf{Rev}) \,;\, \mathbf{Eql} \tag{29}$$

$$\begin{aligned}\mathbf{Pal} = {} & \mathbf{1}_{\mathcal{L}^1} \,;\, \mathbf{True} \,+ \\ & \mathbf{1}_{\mathcal{L}^*-\mathcal{L}^1} \,;\, \mathbf{2} \,;\, (\mathbf{2} \,;\, (\mathbf{Hd} \times \mathbf{Lst}) \times \mathbf{Md}) \,;\, (\mathbf{Smc} \times \mathbf{Pal}) \,;\, \mathbf{And}\end{aligned} \tag{30}$$

where, **Pal**, **Rev**, **Eql**, **True**, **Hd**, **Lst**, **Md**, **Smc** and **And**, are *problems* whose *conditions* are the relations *pal*, *rev*, *eql*, *true*, *hd*, *lst*, *md*, *smc* and *and*, respectively.

Analyzing (29) we realize that, even though *;* and $\times$ are algorithmic operations, we cannot, except in special cases of software reusing, hope that **Rev** and **Eql** will have meaning in our target language. By *target language* we mean the language that our *target machine* (the piece of hardware and software at our disposal) will directly interpret. On the other hand, expression (30) involves only problems such as **True**, **Hd**, **Lst**, **Md**, **Smc**, etc., which we assume our target machine will directly interpret.

So, aside from complexity issues, the purpose of deriving is twofold. The goal of a derivation is a term on problems whose operations are algorithmic and whose symbols have direct meaning for the target machine. The set of symbols on *problems* with meaning depends on the target machine at hand. (One can imagine that in some software engineering environment, i.e., target machine, **Rev** and **Eql** may have meaning, and so, no derivation is needed.) Thus, this set, called the set of *easy problems*, depends on the target machine - as well as, perhaps, some other considerations - and defines the goal set of a derivation.

The preceding discussion has paved the way for some general remarks concerning specification and programming languages. A programming language is required to be executable, preferably in an efficient manner. This is why non-algorithmic operations like difference are not expected to be part of such a language. Also, Hoare and Jifeng [Hoa86] exclude intersection from their programming language mainly for reasons of efficiency. On the other hand, a specification language is not required to be executable. Its prime features should rather be expressiveness and ease of expression. Roughly speaking, the more restricted (but still universal) a programming language is, the more likely it is to attain its goals; whereas a very broad specification language (say, in the spirit of CIP's wide-spectrum language) is more likely to achieve its goals.

Some reasonable candidates for derivation languages are Horn clauses and Martin-Löf's Intuitionistic Type Theory, but both fail to be completely adequate, not from the programing viewpoint but from the specification standpoint. The former falls short of first-order logic as far as expressiveness is concerned. In using the latter to describe problems, one ends up specifying their solutions in an algorithmic manner [Haeu88].

In our case, the specification language is the language of problem-theoretic terms, whereas the programming language is its sub-language consisting only of terms without any non-algorithmic operations. So, our framework, based on the single unifying concept of partial binary relations, supports a truly coherent wide-spectrum language. Of course, each particular application will have a set of *primitive problems* that do not require specification, thereby defining the *application specification language*. This is similar to the characterization of the target language by means of the *problems* that are directly executable by the target machine in question.

8.2. Specification of Problems

Let us now briefly comment on and illustrate the use of our *problem-theoretic language* as a language for specifying *problems* as well as programs.

First of all, notice that in view of Theorems 6.3 and 6.4 we have the full expressive power of first-order logic at our disposal. Thus, we can describe by means of a problem-theoretic term any *problem* originally given by a first-order presentation.

It should now be apparent that we can likewise specify properties of data types. For instance, a simple property of lists is **head**(**cons**(c, x)) = c. One way to state this property within our language is simply $\mathbf{Cns}\ ;\ \mathbf{Hd} = \Pi_1$. This is not quite right (the correct equation should be $(\mathbf{1}_{\mathcal{L}^*} \times \mathbf{1}_{\mathcal{L}^*})\ ;\ \mathbf{Cns}\ ;\ \mathbf{Hd} = \Pi_1\ ;\ \mathbf{1}_{\mathcal{C}}$); but is already quite useful whenever we wish to use the above property in the context of a given expression - for then the context will take care of the identities corresponding to the carriers. In other words, properties of abstract data types are expressed by equations with explicit carrier indication, but the latter is not necessary within a context, when it becomes just "coupling" information and can be discarded.

Having settled the question of expressive power, the next natural question is " Is it easy to express what one wishes? " As it might be expected, the answer depends on having a certain knack and gestalt of the language at hand. The lack of variables may at first seem bothersome, but it has its own features. For instance, let **Un** be the *problem* of obtaining the union of two sets. We can express the commutativiy of union simply by

$$\mathbf{Un} = (\Pi_2 \nabla \Pi_1)\ ;\ \mathbf{Un},$$

or, more conspicuously, by

$$(\Pi_1 \nabla \Pi_2)\ ;\ \mathbf{Un} = (\Pi_2 \nabla \Pi_1)\ ;\ \mathbf{Un}.$$

8.3. Some Programming Aspects

Recall, from Section 6, that a structure $\mathfrak{B}$ assigns to a variable-free *problem-theoretic* term T the problem $\mathfrak{B}$[T] denoted by it. Now, a *target machine* can be regarded as a special structure, in the sense that it is able to directly interpret the *easy problems.*

Indeed, consider a *target machine* H and let $\mathcal{F}$ be a set of symbols corresponding to the *easy problems*. Consider also a set $\mathcal{W}$ of variables over *problems*. By an *algorithmic term* (for

H) we mean a term built from $\mathcal{F} \cup \mathcal{W}$ by means of the *algorithmic operation symbols*. Our *target programming language* is the set of such *algorithmic terms*, i. e., the closure $\mathcal{A}(\mathcal{W})$ of $\mathcal{F} \cup \mathcal{W}$ under the *algorithmic operation symbols*.

Now, we may consider a *target machine* H as a function $H : \mathcal{F} \to \mathbb{P}\mathcal{U}$ assigning to each symbol $S \in \mathcal{F}$ a *problem* H[S]. We can extend H to a function $\hat{H}: \mathcal{A}(\emptyset) \to \mathbb{P}\mathcal{U}$, where $\hat{H}[P]$ is defined by induction as:

$\hat{H}[S] = H[S]$ if $S \in \mathcal{F}$
$\hat{H}[P + Q] = \hat{H}[P] + \hat{H}[Q]$
$\hat{H}[P ; Q] = \hat{H}[P] ; \hat{H}[Q]$
$\hat{H}[P \times Q] = \hat{H}[P] \times \hat{H}[Q]$
. . . .

Thus, $\hat{H}[P]$ is the machine interpretation of P for $P \in \mathcal{A}(\emptyset)$.

Let us now examine the effect of introducing variables over *problems*. We may start by extending $H : \mathcal{F} \to \mathbb{P}\mathcal{U}$ to $H_{\mathcal{W}}: \mathcal{F} \cup \mathcal{W} \to \mathbb{P}\mathcal{U}$ by assigning to each variable $W \in \mathcal{W}$ the problem **0**. We can now extend $H_{\mathcal{W}}$ to a function $\hat{H}_{\mathcal{W}}: \mathcal{A}(\mathcal{W}) \to \mathbb{P}\mathcal{U}$ as above.

Consider the expression $X = T(X)$, where $T(X) \in \mathcal{A}(\mathcal{W})$ is an *algorithmic term* on *problems*. We can regard, intuitively, such expression as defining a *problem* if, by successive *unfoldings*, each data of its *data carrier* is connected to a result in its *result carrier* by a "chain" of *easy problems*.

As usual, we may view *unfoldings* as iterated substitutions. For instance, for the expression T(X), $\mathit{unfold}(T(X)) = \{T(X), T(X/T), T(X/T(X/T)), T(X/T(X/T(X/T))),.\}$. Formally, if $\sigma : X \to T(X)$, is a substitution, $\mathit{unfold}(\sigma) = \{\sigma(X), \sigma^2(X), \sigma^3(X), \ldots\}$. We can now further extend $\hat{H}_{\mathcal{W}}$ to assign a computational meaning to T(X) as follows

$$\hat{H}_{\mathcal{W}}[T(X)] = \sum\{\hat{H}_{\mathcal{W}}[P] : P \in \mathit{unfold}(X = T)\} \tag{31}$$

The justification for this relies on the fact that the *algorithmic operations* are monotonic (and continuous) on $\langle \mathbb{P}\mathcal{U}, \subseteq \rangle$ and $\langle \mathbb{P}\mathcal{U}, \subseteq \rangle$ is a *complete lattice*. Thus, from a well-known theorem of Tarski's [Tar55], every expression of the form T(X) will have a *fixpoint*, and thus a meaning. We should observe that (31) is just an expression for the *least fixpoint* because of the continuity of the *algorithmic operations* with respect to $\subseteq$ [Sco72].

On the other hand, once H has successfully assigned *problem* **P** to the symbol P, we may consider a new *virtual target machine* (H|P) which extends H by assigning **P** to P.

Another question that should be addressed is the termination of an expression of the form $X = T(X)$. For instance, when will expression (30) terminate?

We should recall that in Section 3 we claimed that expression (11) for the reversal of a list captures part of the inductive argument displayed there; namely, that *rev* splits recursively a list till decomposing it completely into elements of $\mathcal{L}^1$, i.e., simple data for the list reversal problem. In reviewing this claim for expression (30), we should notice that the full expression of this inductive argument consists of (30) together with the expression $\mathbf{1}_{\mathcal{L}^*} ; \widetilde{\mathbf{Md}}^{;*} = \mathbf{1}_{\mathcal{L}^*}$, the latter being a way of expressing termination in our calculus. It is very important to point out that such expressions carry the inductive argument from the metalanguage into the calculus. In fact, the termination condition for a simple divide-and-conquer schema like $\mathbf{P} = \mathbf{1}_{\mathbf{Easy}} ; \mathbf{Easy} + \mathbf{Split} \; ; \mathbf{P} ; \mathbf{Join}$ is easily seen to be $\mathbf{1}_{\mathbf{Easy}} ; \widetilde{\mathbf{Split}}^{;*} = \mathbf{1}_{\mathbf{P}}$.

8.4. Program Derivation

We shall now give some indications concerning the appropriateness of our *problem-theoretic language* and *calculus* for deriving and reasoning about programs. We shall concentrate our remarks on the expression of strategies and design decisions, as well as their manipulations.

Typically, the specification of the input-output behavior of a program consists of a precondition $\phi(x)$ and a postcondition $\psi(x, y)$, and the latter can always be written as $\psi(x, y) \wedge \theta(y)$. Now, if $\mathbf{1}_\phi$ and $\mathbf{1}_\theta$ are terms denoting the identities over the sets defined by $\phi(x)$ and $\theta(y)$, respectively, then our programming problem can be described by the term $\mathbf{1}_\phi ; \Psi ; \mathbf{1}_\theta$, where Ψ is a term denoting the *problem* whose *condition* is the relation defined by $\psi(x, y)$. This specification can be regarded as a *generate-and-test* procedure, where $\mathbf{1}_\phi ; \Psi$ is *generate* and the filter $\mathbf{1}_\theta$ is *test.* In order to derive more reasonable algorithms, some strategies are useful.

We have already seen some simple strategies in Section 3. The expression of some general strategies in our problem-theoretic formalism can be as follows (some of the names are not standard).

Case division : $\mathbf{P} = \mathbf{Q}' + \mathbf{Q}''$
Decomposition : $\mathbf{P} = \mathbf{Q}' \times \mathbf{Q}''$
Interpolation : $\mathbf{P} = \mathbf{Q}' ; \mathbf{Q}''$
Reduction : $\mathbf{P} = \mathbf{T} ; \mathbf{Q}' ; \mathbf{R}$
Reduction to decomposition: $\mathbf{P} = \mathbf{T} ; (\mathbf{Q}' \times \mathbf{Q}'') ; \mathbf{R}$
n-ary Divide-and-Conquer: $\mathbf{P} = \mathbf{E} + \mathbf{S} ; (\mathbf{P}^{\times n}) ; \mathbf{J}$.

The derivation in Section 3 has started with the application of the strategies of case division and trivialization. It then proceeded by the introduction of the eureka

$$\mathbf{Erk} = \widetilde{\mathbf{3} ; (\mathbf{Lhd} \times \mathbf{Md} \times \mathbf{Llst})},$$

and the use of a reasoning akin to *weakest prespecification* to solve the equation $X ; \mathbf{Erk} = \mathbf{Rev}_1$.

At the end of Section 3 we indicated an alternative derivation for the *palindrome problem*, namely by employing a strategy of a more algebraic flavor. It can be expressed as

$$\mathbf{1}_{\mathcal{L}^*-\mathcal{L}^1} = \mathbf{Dcmp} ; \mathbf{Rcmb} \text{ with } \mathbf{Dcmp} = \mathbf{3} ; (\mathbf{Lhd} \times \mathbf{Md} \times \mathbf{Llst}).$$

From this we can derive within our calculus an expression for **Rcmb**, namely

$$\mathbf{Rcmb} = \widetilde{\mathbf{3} ; (\mathbf{Lhd} \times \mathbf{Md} \times \mathbf{Llst})}.$$

This expression does not make explicit the deterministic nature of **Rcmb**. Nevertheless, by making use of properties of lists, such as $\widetilde{\mathbf{Cns}} = \mathbf{Hd} \nabla \mathbf{Llst}$, we can derive the explicitly deterministic algorithm

$$\mathbf{Rcmb} = ((\Pi_1 \nabla \Pi_2 ; \Pi_1) \nabla \Pi_2 ; \Pi_2) ; (\mathbf{Cns} ; \mathbf{App}).$$

As a matter of fact, our calculus allows one to take advantage of the flexibility afforded by nondeterministic expressions. For instance, consider the following expression for the identity over non-ordered lists:

$$\begin{aligned}\mathbf{1}_{\neg\mathrm{Ord}} = {} & \widetilde{\mathbf{Rrng} ; (\mathbf{App} \times \mathbf{Cns}) ; \mathbf{Cnc}} ; (\mathbf{1}_{\mathcal{L}^*} \times \mathbf{1}_{\neg o} \times \mathbf{1}_{\mathcal{L}^*}) ; \\ & ; \mathbf{Rrng} ; (\mathbf{App} \times \mathbf{Cns}) ; \mathbf{Cnc}\end{aligned}$$

with the *formal noise* term $\mathbf{Rrng} = ((\Pi_1 \nabla \Pi_2 ; \Pi_1 ; \Pi_1) \nabla (\Pi_2 ; \Pi_1 ; \Pi_2 \nabla \Pi_2 ; \Pi_1))$ taking care of the rearranging (necessary because of the non-associativity), where $\mathbf{1}_{\neg o}$ is the identity over pairs of elements not satisfying order o, whereas **Cnc** is the *problem* corresponding to concatenation of lists. From this non-deterministic expression, one can derive any sorting algorithm based on the *sorting-by-inversion* strategy. Indeed, particular deterministic implementations of $\widetilde{(\mathbf{App} \times \mathbf{Cns}) ; \mathbf{Cnc}}$, will yield shakersort and the various versions of bubblesort.

9. CONCLUSIONS

We have presented a formalism based on partial binary relations, rich enough to support a calculus for program derivation. The algebraic flavor of the calculus is adequate for program derivation because one can express specifications, programs, eurekas, strategies and local decisions on the desired level of abstraction. It extends the calculi of Tarski and Hoare, and hence, Dijkstra's weakest precondition approach [Dij76] as well as VDM. It is important to notice that, since termination can be expressed and manipulated within the calculus, totally correct programs can be derived by solving a system of algebraic equations as indicated in Sections 3 and 8.

We have proved that the relational semantics of any program derivation calculus associating preconditions to programs must go beyond Tarski's relations into the realm of partial relations (Theorems 4.4 and 4.7). This corroborates some intuitive feelings about the oversimplification forced by the confinement to total relations.

We have also proved that the expressive power of our calculus encompasses that of first-order logic, by showing how it can express Tarski's definition of satisfaction (Theorem 6.2). The finitary part of our calculus is, thus, equivalent to first-order logic. In particular, we have a new candidate for an algebra of logic and complete axiomatizations for our calculus can be constructed. By means of the closure operations we can express non-first-order properties, such as termination.

Our framework is general enough to deal with problems on problems, general strategies, such as reduction, etc. We do not run into circularity problems because we rely on our model-theoretic development based on set theory (Section 5).

We employ operations on problems in order to have terms, that can be regarded as (possibly nondeterministic) programs. Also, the nondeterminism within the derivation process appears in the form of inequations, in addition to equations. This amounts to relations on problems.

The long-range goal of our present development is an algebraic calculus whose terms can capture, not only effectiveness, but also complexity and testability. We also plan to extend our problem-theoretic approach to cover the entire software process, from requirements to programs [Hae89, 90b]. The relational approach seems adequate to capture ideas related to single-case propensities [Han81] and modern theories of induction [Fri90] while program testing and specification elicitation (like theory construction) are related to inductive inference [Ang83].

Even though our problem-theoretic formalism is based on partial binary relations, it provides not merely another specification/programming language or derivation calculus, but rather a framework for expressing, and reasoning about, the semantics of specification and programming languages. In this sense, our considerations are perfectly general.

REFERENCES

[Ang83] Angluin, D., Smith, C. H., "Inductive Inference: Theory and Methods", *Computing Surveys* **15** (1983), pp. 237-269.

[Bac90] Backhouse, R. C., Bruin P.,J., Malcom, G., "A Relational Theory of Types", *Document* **638-BUR-5**, 41st Meeting of the IFIP Working Group 2.1."Programming Languages and Calculi"; Chester, England, 1990.

[Bur80] Burris, D., Sankappanavar, G., *A Course in Universal Algebra*, Springer-Verlag, New York, 1980.

[Chi51] Chin, L. H.,Tarski, A., "Distributive and Modular Laws in the Arithmetic of Relation Algebras"; *Univ. of California Publications in Mathematics, New Series* **1.9** (1951), pp. 341-384.

[deB72] de Bakker, J. W., de Roever, W. P., "A Calculus for Recursive Program Schemes". Nivat, M. (ed.), *Proc. IRIA Symp. on Automata, Formal Languages and Programming*, North Holland, Amsterdam, 1972, pp. 167-196.

[Dij76] Dijkstra, E. W., *A Discipline of Programming*, Prentice Hall, Englewood Cliffs, 1976.

[Ebb80] Ebbinghaus, H. D., Flum, J., Thomas, W., *Mathematical Logic*, Springer-Verlag, New York, 1980.

[End72] Enderton, H. B., *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.

[Fre79] Frege, G., "Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens", Louis Nebert, Halle, 1879 {English translation in van Heijenoort, J. (ed.), *From Frege to Gödel*, Harvard Univ. Press, Cambridge, 1967, pp. 1-82}.

[Fri90] Friedman, K. S., *Predictive Simplicity: Induction Exhum'd*, Pergamon Press, Oxford, 1990.

[Grä71] Grätzer, G. D., *Lattice Theory*, W. H. Freeman, San Francisco, 1971.

[Hae87] Haeberer, A. M., Baum G., Veloso P.,A.,S., "On an Algebraic Theory of Problems and Software Development"; *Pont. Universidade Católica, Res. Rept.* **MCC 2/87** (1987).

[Hae89] Haeberer, A. M., Veloso P.,A.,S., Baum G., *Formalización del Proceso de Desarrollo de Software*, Kapeluz, Buenos Aires, 1989.

[Hae90a] Haeberer, A. M., Veloso P.,A.,S., Elustondo P., "Towards a Relational Calculus for Software Construction", *Document* **640-BUR-5**, 41st Meeting of the IFIP Working Group 2.1."Programming Languages and Calculi"; Chester, England, 1990 {Previously appeared as "Towards a Problem-Oriented Relational Calculus for

Software Construction"; *Pont. Universidade Católica, Res. Rept.* **MCC 19/89**, 1989}.

[Hae90b] Haeberer, A. M., Veloso P.,A.,S., "Why Software Development is Inherently Non-Monotonic: A Formal Justification". Trappl R. (Ed.), *Cybernetics and Systems Research*, World Publ. Corp., London, 1990, pp. 51-58.

[Haeu88] Haeusler, E. H., "Intuitionistic Type Theory and General Problem Theory: a relationship"; *Pont. Universidade Católica, Res. Rept.* **MCC 17/88**, 1988.

[Han81] Hanna, J. F., "Single Case Propensities and the Explanation of Particular Events", *Synthese* **48** (1981), pp. 409-436.

[Hoa86] Hoare, C. A. R., Jifeng He, "The Weakest Prespecification", *Fundamenta Informaticae* **9** (1986), pp. 51-84, 217-252.

[Jon80] Jones, C., *Software Development: A Rigorous Approach*, Prentice Hall, Englewood Cliffs, 1980.

[Jón52] Jónsson, B.,Tarski, A., "Boolean Algebras with Operators", *Amer. J. of Math.* **74** (1952), pp. 127-162.

[McK40] McKinsey, J. C. C., "Postulates for the Calculus of Binary Relations", *J. of Symbolic Logic* **5.3** (1940), pp. 85-97.

[Mar84] Martin-Löf, P., *Intuitionistic Type Theory*, Bibliopolis, Napoli, 1984

[Par90] Partsch, H. A., *Specification and Transformation of Programs*, Springer-Verlag, Berlin, 1990.

[Pol57] Polya, G., *How to Solve it: A New Aspect of the Mathematical Method*, Princeton Univ. Press, Princeton, 1957.

[Sch85] Schmidt, G., Ströhlein, T., "Relation Algebras: concepts of points and representability", *Discrete Mathematics* **54** (1985), pp. 83-92.

[Sch95] Schröder, E., *Vorlesungen über die Algebra der Logik*, Leipzig; i, 1890; ii, part 1, 1891; ii part 2, 1905; iii, part 1, 1895 {reprinted by Chelsea, New York, 1966}.

[Sco72] Scott, D., "Lattice Theory, Data types and Semantics", Rustin, R. (Ed.), *Formal Semantics of Programming Languages*, Prentice Hall, Englewood Cliffs, 1972, pp. 65-106.

[Smi83] Smith, D R., "A Problem Reduction Approach to Program Synthesis". Proc. Int. Joint Conf. on Artificial Intelligence, Karlsruhe, 1983, pp. 32-36.

[Sup60] Suppes, P., *Axiomatic Set Theory*, Van Nostrand, Princeton, 1960.

[Tar41] Tarski, A., "On the Calculus of Relations", *J. of Symbolic Logic* **6.3** (1941), pp. 73-89.

[Tar55] Tarski, A., "A Lattice-Theoretical Fixpoint Theorem and its Applications", *Pacific Journal of Mathematics.* **5** (1955), pp. 285-309.

[Vel81] Veloso, P. A. S., Veloso, S. R. M., "Problem Decomposition and Reduction: Applicability, Soundness, Completeness". Trappl R. , Klir J. , Pichler F. (Eds.), *Progress in Cybernetics and Systems Research,* **8**, Hemisphere, Washington, 1981, pp. 199-203.

[Vel84] Veloso, P. A. S., "Outlines of a General Theory of General Problems", *Philosophia Naturalis* **21.2-4** (1984), pp. 354-365.

[Vel90] Veloso, P. A. S., Haeberer, A. M. "Towards a New Algebra of Firs-Order Logic", *Pont. Universidade Católica; Res. Rept.* **MCC 18/90** (1990).

ACKNOWLEDGMENTS

Partial financial support from IBM Brasil and the Brazilian agencies CNPq and FAPERJ is gratefully acknowledged. The editor and the referees are also to be thanked for their helpful comments and criticisms to an earlier version of this paper.