PUC

Series: Monografias em Ciência da Computação,
No. 19/89

TOWARDS A PROBLEM-ORIENTED RELATIONAL CALCULUS FOR SOFTWARE CONSTRUCTION

Armando M. Haeberer
Paulo A. S. Veloso
Pablo M. Elustondo

Departamento de Informática

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

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC RIO, Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
BRASIL

Tel.:(021)529-9386 TELEX:31078 FAX:(021)274-4546
BITNET:userrtlc@lncc.bitnet

PUC-RJ - Departamento de Informática

Series : Monografias em Ciência da Computação No. 19 / 89

Series Editor : Paulo A. S. Veloso September 1989

TOWARDS A PROBLEM - ORIENTED RELATIONAL CALCULUS FOR SOFTWARE CONSTRUCTION +

Armando M. Haeberer

Paulo A. S. Veloso

Pablo Elustondo *

* Assistant researcher of the ETHOS project

+ Research partly sponsored by the ETHOS project of the Argentine-Brazilian Program of Research and Advanced Studies in Computer Science and by FINEP.

ABSTRACT

A calculus for program derivation is presented. This calculus is based on the algebraic theory of problems. A problem consists of two sets and a relation between them. The calculus is based on properties of problems as atomic objects.

Several levels of detail can be uniformly handled within the calculus : from global strategies through design decisions to local manipulations.

An important feature of the calculus is its ability to capture synthetic aspects. This is due to the design of the calculus aiming at its extension to cover the entire software process.

Key words : Software development, program derivation, program construction, strategies, decomposition, program design, software construction, algebra of relations, algebra of problems.

RESUMO

Apresenta-se um cálculo, baseado na teoria algébrica de problemas, para derivação de programas. Um problema consiste de dois conjuntos e de uma relação entre eles.
O cálculo se baseia em propriedades de problemas como objetos atômicos. Vários níveis de detalhe podem ser uniformemente tratados no cálculo : desde estratégias globais passando por decisões de projeto até manipulações locais.
Uma característica importante do cálculo é a possibilidade de captar aspectos sintéticos, característica esta que se deve à sua própria construção, visando a sua extensão para cobrir todo o processo de software.

Palavras chave : Desenvolvimento de software, derivação de programas, construção de programas, estratégias, decomposição, projeto de programas, construção de software, álgebra de relações, álgebra de problemas.

RESUMEN

Se presenta un cálculo, basado en la teoría algebraica de problemas, para la derivación de programas. Un problema consiste de dos conjuntos y de una relación entre ellos. El cálculo se basa en propiedades de problemas como objetos atómicos.
Varios niveles de detalle pueden ser uniformemente tratados en el cálculo : desde estrategias globales pasando por decisiones de diseño hasta manipulaciones locales.
Una característica importante del cálculo es la posibilidad de captar aspectos sintéticos, característica esta que se debe a su propio diseño, con vistas a su extensión para cubrir todo el proceso de software.

Palabras clave : Desarrollo de software, derivación de programas, construcción de programas, estrategias, descomposición, diseño de programas, construcción de software, álgebra de relaciones, álgebra de problemas.

I. INTRODUCTION AND MOTIVATIONS.

Recent years have witnessed a series of efforts attemping to model the software development process. These efforts range from the search for a canonical step [Leh84; LST84; L+B85; M+T84] through calculi for program design [Sin85; Sin86; JHW86]. Software development process consists not only of program construction but also of specification construction. The process of specification construction starts from the actual problem and obtains a formal description for it in a precise formalism. Its correctness criterion is a synthetic one, which specification validation aims to achieve. The process of program construction, which starts from a formal specification of a problem and obtains a program for it, has an analytic correctness criterion, relying on proofs.

The entire software development process, which starts from the actual problems and obtains a program for it, involves a correctness criterion which is not factorizable, i.e. it is neither purely synthetic nor purely analytic. As stated by Turski [T+M87] and formally proved in [H+V89b], the synthetic character of specification construction sneaks into the correctness criterion of program construction. So, one might be led into considering a purely empirical criterion for the whole process. But, the relation of "being a program for an actual problem" is a complex disposition, involving arguments that are non-refutable in principle. These arguments must be subsumed under their theoretical counterparts. As a result, establishing the correctness of the entire process involves an inevitable intertwining of experiments and proofs [H+V89b].

We would like to have a uniform formalism for reasoning about the entire software development process. In other words, we do not want to deal with proofs in the language and experiments in the meta-language. Such a formalism should be able to deal with synthetic aspects within it, which leads to its non-monotonicity [H+V89a]. This is what happens when one applies a formalism to an empirical science.

We have two kinds of desiderata for a calculus for software construction, namely, those imposed by the nature of the software process per se and those expected in a good reasoning tool. Among the former, we have the ability to cope with non-monotonicity and other aspects due to the synthetic character of the software process. As for the latter, an important desideratum concerns the uniformity of the framework; especially, uniformity with respect to:

. the objects of different phases: problems, specifications and programs;

. the nature of the obligations: proofs and validations;

. levels: programs, families of programs, programming methods, etc.;

. size and complexity: programming in the small and programming in the large.

We have barely started the development of our calculus. Its central concept is that of *(relational) problem*, introduced in [Vel84] and further developed in

[HBV87; V+H89a]. This concept was used to model several aspects of the software process in an exploratory way [HVB89; V+H89b]. Some preliminary steps towards our program derivation calculus appear in [EVHV89; V+E89]. Finally we used the problem-theoretic framework together with Carnap's ideas to prove the above results on non-factorizability and non-monotonicity [H+V89a; 89b].

In this paper we shall emphasize our .calculus for program construction, which is meant to be part of a calculus for software construction.

In constructing a program from a formal specification, one deals with diverse levels of details and, hence, of decisions. For instance, an early decision concerning the application of a particular divide-and-conquer technique will have the status of a strategy, whereas an application of unfolding has some flavor of a local manipulation.

Another aspect is the right amount of details in the manipulations. This is enhanced by the possibility of hiding undesired details at each level of abstraction. In other words, one should be able to choose the appropriate atomic objects at each step.

Our program construction calculus should cope with both aspects above: we do not want our notation to bury strategic decisions under local manipulations or to force us to confuse different levels of abstraction.

We would like to be able to manipulate descriptions of problems and programs much as one does with algebraic expressions. This is why we have decided to base our calculus on a few basic algebraic laws.

II. PROBLEMS.

II.1. BASIC DEFINITIONS.

II.1.1. **Definition.** A *problem* $\mathbf{P}$ over a universe $\mathbb{U}$ is a 3-tuple of sets $\mathbf{P}=\langle D, R, \rho\rangle$ such that:

i. $D, R \subseteq \mathbb{U}$

ii. $\rho \subseteq D \times R$.

Notation. We will use subscript $\mathbf{P}$ to indicate that $\mathbf{P}$ is the problem we are referring to. So:

$$D_{\mathbf{P}} = \pi_1(\mathbf{P})$$

$$R_{\mathbf{P}} = \pi_2(\mathbf{P})$$

$$\rho = \pi_3(\mathbf{P})$$

where π_i is the ith projection function.

We will say that $D_{\mathbf{P}}$ is the *data domain*, $R_{\mathbf{P}}$ the *result domain* and ρ the *condition* of problem $\mathbf{P}$.

Henceforth we shall fix a universe $\mathbb{U}$ and denote by $\mathbb{P}$ the set of all problems over $\mathbb{U}$.

II.1.2. **Definition**. Two problems $\mathbf{P}$ and $\mathbf{Q}$ are equal iff they are equal as 3-tuples, i.e.

$$\mathbf{P} = \mathbf{Q} \leftrightarrow D_{\mathbf{P}} = D_{\mathbf{Q}} \wedge R_{\mathbf{P}} = R_{\mathbf{Q}} \wedge \mathcal{P} = \mathcal{Q} .$$

II.1.3. **Definition**. A problem $\mathbf{P}$ is *viable* iff every element of $D_{\mathbf{P}}$ is related to at least one element of $R_{\mathbf{P}}$ by the relation $\mathcal{P}$, i.e.

$$\mathbf{P} \text{ is } \textit{viable} \leftrightarrow (\forall d)[d \in D_{\mathbf{P}} \rightarrow (\exists r)(r \in R_{\mathbf{Q}} \wedge \mathcal{P}(d, r))].$$

Notice that $\mathbf{P}$ is *viable* iff $D_{\mathbf{P}} = \mathcal{Dom}(\mathcal{P})$[1].

We denote by $\mathbb{V}$ the set of all viable problems over $\mathbb{U}$ (obviously, $\mathbb{V} \subseteq \mathbb{P}$).

II.1.4. **Definition**. A problem $\mathbf{P}$ is:

i. *surjective* iff $R_{\mathbf{P}} = \mathcal{Ran}(\mathcal{P})$

ii. *deterministic* iff $\langle a, b\rangle \in \mathcal{P} \wedge \langle a, c\rangle \in \mathcal{P} \rightarrow b=c$,

iii. *injective* iff $\langle a, b\rangle \in \mathcal{P} \wedge \langle c, b\rangle \in \mathcal{P} \rightarrow a=c$.

We say that a problem is *gentle* iff it is viable and surjective. We denote by $\mathbb{G}$ the set of all gentle problems over $\mathbb{U}$ (obviously, $\mathbb{G} \subseteq \mathbb{V}$). It is also clear that the data and result domains of a gentle problem equal, respectively, the domain and range of its condition; so, we can refer to a gentle problem simply by its condition, i.e. $\mathcal{P} = \langle \mathcal{Dom}(\mathcal{P}), \mathcal{Ran}(\mathcal{P}), \mathcal{P}\rangle$.

II.2. SPECIAL PROBLEMS.

II.2.1. Identities.

II.2.1.1. **Definition**. We denote by $\mathbb{I}$ the family of all viable problems over $\mathbb{U}$ whose condition is an identity relation, i.e.

$$\mathbb{I} = \{\langle A, A, \mathfrak{I}_A\rangle : A \subseteq \mathbb{U}\}$$

where $\mathfrak{I}_A$ is the identity relation over A.

II.2.1.2. **Definition**.

i. $\mathbf{1}_{\mathbf{P}} = \langle D_{\mathbf{P}}, D_{\mathbf{P}}, \mathfrak{I}_{D_{\mathbf{P}}}\rangle$.

ii. $\mathbf{1}^{\mathbf{P}} = \langle R_{\mathbf{P}}, R_{\mathbf{P}}, \mathfrak{I}_{R_{\mathbf{P}}}\rangle$.

Notice that $\mathbf{1}_{\mathbf{P}} \in \mathbb{G}$ and $\mathbf{1}^{\mathbf{P}} \in \mathbb{G}$.We will use $\mathbf{1}$ polymorphically to refer to any problem belonging to $\mathbb{I}$.

[1]We will use along this paper the usual notation $\mathcal{Dom}$ and $\mathcal{Ran}$ respectively for the domain and range of a relation.

II.2.1.3. **Theorem.** For any $\mathbf{P} \in \mathbb{P}$ the following are equivalent:

i. $\mathbf{P} \in \mathbb{I}$

ii. $\mathbf{1_P} = \mathbf{P}$

iii. $\mathbf{1^P} = \mathbf{P}$.

II.2.2. Extremes.

II.2.2.1. **Definition.**

i. $\mathbf{0} = \langle \varnothing, \varnothing, \varnothing \rangle$.

ii. $\mathbf{0_P} = \langle D_\mathbf{P}, \varnothing, \varnothing \rangle$.

iii. $\mathbf{0^P} = \langle \varnothing, R_\mathbf{P}, \varnothing \rangle$.

iv. $\mathbf{\underset{P}{0}} = \langle D_\mathbf{P}, R_\mathbf{P}, \varnothing \rangle$.

Notice that $\mathbf{\underset{P}{0}}$ is not a viable problem. We will denote by $\mathbb{0}$ the family of all the problems of the form $\langle A, B, \varnothing \rangle$ for $A, B \subseteq \mathbb{U}$.

II.2.2.2. **Definition**

$\infty = \langle \mathbb{U}, \mathbb{U}, \mathbb{U} \times \mathbb{U} \rangle$

II.2.3. Forks.

II.2.3.1. **Definition.** Given a problem $\mathbf{P}$ we define the *fork* problems $\mathbf{2_P}$, $\mathbf{3_P}$, etc., of $\mathbf{P}$ as:

i. $\mathbf{2_P} = \langle D_\mathbf{P}, \{\langle y, y \rangle : y \in D_\mathbf{P}\}, \{\langle x, \langle x, x \rangle\rangle : x \in D_\mathbf{P}\}\rangle$,

ii. $\mathbf{3_P} = \langle D_\mathbf{P}, \{\langle y, y, y \rangle : y \in D_\mathbf{P}\}, \{\langle x, \langle x, x, x \rangle\rangle : x \in D_\mathbf{P}\}\rangle$,

iii. ...

Notice that problems $\mathbf{2_P}$, $\mathbf{3_P}$, etc., are gentle.

We denote by $\mathbb{2}$, the family of all problems of the form $\mathbf{2_P}$ for $\mathbf{P} \in \mathbb{P}$. Similarly for $\mathbb{3}$, etc.

II.2.4. Projections.

II.2.4.1. **Definition.** By *ith-projection* (denoted by $\mathbf{\Pi_i}$) we mean any representant of the equivalence class of problems such:

i. D_{Π_i} is a set of n-tuples, for some positive integer $n \geq i$, of elements of the universe,

ii. R_{Π_i} $\{y : x \in D_{\Pi_i} \wedge y = \pi_i(x)\}$, and

iii. $q_{\Pi_i} = \{\langle x, y \rangle : x \in D_{\Pi_i} \wedge y \in R_{\Pi_i} \wedge y = \pi_i(x)\}$.

Notice that every problem $\mathbf{\Pi_i}$ for any choice of a positive integer i is gentle.

II.3. ALGORITHMIC OPERATIONS.

II.3.1. Sum of Problems.

II.3.1.1. **Definition**. Given problems **P** and **Q**, we define their *sum* as the problem **P+Q** whose components are the union of the components of **P** and **Q**, i.e.

$$D_{P+Q} = D_P \cup D_Q, R_{P+Q} = R_P \cup R_Q, \mathcal{P+Q} = \mathcal{P} \cup \mathcal{Q}.$$

II.3.1.2. **Theorem**. Properties of the sum:

i.	Associativity	$(P+Q)+R=P+(Q+R)$
ii.	Commutativity	$P+Q=Q+P$
iii.	Idempotence	$P+P=P$
iv.	Neuter	$P+0=P$
v.	Zero	$P+\infty=\infty$

Notice that given a problem **P**, $P+\frac{0}{P}=P$. So, $\frac{0}{P}$ is a neuter for any problem **Q** differing from **P** only in its condition.

II.3.1.3. **Definition**. Given a family $\mathcal{F}$ of problems we define their *summation* as the problem:

$$\Sigma_{\mathcal{F}}X = \langle \bigcup_{\mathcal{F}} D_X, \bigcup_{\mathcal{F}} R_X, \bigcup_{\mathcal{F}} q_X \rangle$$

where $\bigcup_{\mathcal{F}}$ stands for the generalized union over $\mathcal{F}$.

II.3.2. Product of Problems.

II.3.2.1. **Definition**. Given the problems **P** and **Q**, we define their *product* **P∘Q** as the problem whose components are:

$$D_{P\circ Q} = D_P, R_{P\circ Q} = R_Q, \mathcal{P}\circ\mathcal{Q} = \mathcal{P}/\mathcal{Q}$$

where / stands for the relative product of relations.

II.3.2.2. **Theorem**. Properties of the product:

i. Associativity

$(P\circ Q)\circ R=P\circ(Q\circ R)$

ii. Left distributivity over sum

$P\circ(Q+R)=P\circ Q+P\circ R$

iii. Right distributivity over sum

$(Q+R)\circ P=Q\circ P+R\circ P$

iv.	Left neuter	$1_P\circ P=P$
v.	Right neuter	$P\circ 1^P=P$.
vi.	Bilateral zero	$\frac{0}{P}\circ P = \frac{0}{P} = P\circ\frac{0}{P}$.

II.3.2.3. **Definition**. Given a problem **P**, we will denote by $\mathbf{P}^{*n}$ the product of **P** by itself n times, i.e.

$$\mathbf{P}^{*0} = \mathbf{1}^{\mathbf{P}}$$

$$\mathbf{P}^{*n+1} = \mathbf{P} * \mathbf{P}^{*n}.$$

II.3.3. Direct Product of Problems.

II.3.3.1. **Definition**. Given problems **P** and **Q**, we define their *direct product* $\mathbf{P} \times \mathbf{Q}$ as the problem whose components are:

$D_{\mathbf{P}\times\mathbf{Q}} = D_{\mathbf{P}} \times D_{\mathbf{Q}}$, $R_{\mathbf{P}\times\mathbf{Q}} = R_{\mathbf{P}} \times R_{\mathbf{Q}}$ and

$$\mathcal{P} \times \mathcal{Q} = \{\langle\langle d, d'\rangle, \langle r, r'\rangle\rangle : \langle d, r\rangle \in \mathcal{P} \wedge \langle d', r'\rangle \in \mathcal{Q}\}$$

II.3.3.2. **Theorem**. Properties of the direct product:

i. Left distributivity over sum

$\mathbf{P}\times(\mathbf{Q}+\mathbf{R})=\mathbf{P}\times\mathbf{Q}+\mathbf{P}\times\mathbf{R}$

ii. Right distributivity over sum

$(\mathbf{Q}+\mathbf{R})\times\mathbf{P}=\mathbf{Q}\times\mathbf{P}+\mathbf{R}\times\mathbf{P}$

iii. Zero $\mathbf{P} \times \mathbf{0} = \mathbf{0} \times \mathbf{P} = \mathbf{0}$.

Notice that $\mathbf{P} \times \mathbf{0}_{\mathbf{P}} = \mathbf{0}_{\mathbf{P}} \times \mathbf{P} = \mathbf{0}_{\mathbf{P}\times\mathbf{P}}$.

II.3.3.3. **Theorem**. $\mathbf{P}*\mathbf{Q} \times \mathbf{R}*\mathbf{S} = (\mathbf{P} \times \mathbf{R}) * (\mathbf{Q} \times \mathbf{S})$.

II.3.3.4. **Definition**. Given a family $\mathcal{F}$ of problems we will call *generalized direct product* over $\mathcal{F}$ the problem:

$$\times_{\mathcal{F}}x = \langle \times_{\mathcal{F}}D_x, \times_{\mathcal{F}}R_x, \mathcal{q}\rangle$$

where $\times_{\mathcal{F}}$ stands for the generalized Cartesian product over $\mathcal{F}$ and $\mathcal{q} = \{\langle x, y\rangle \in \times_{\mathcal{F}}D_x \times \times_{\mathcal{F}}R_x : (\forall i \in \mathcal{F})\langle x(i), y(i)\rangle \in q(i)\}$.

Since $\times$ is not associative, we should write $(\mathbf{P} \times \mathbf{Q}) \times \mathbf{R}$ or $\mathbf{P} \times (\mathbf{Q} \times \mathbf{R})$ to avoid ambiguity. Therefore, by $\mathbf{P} \times \mathbf{Q} \times \mathbf{R}$ we will denote, by convention, the generalized direct product over the family $\{\mathbf{P}, \mathbf{Q}, \mathbf{R}\}$.

II.3.3.5. **Definition**. Given a problem **P**, we will denote by $\mathbf{P}^{\times n}$ the problem:

$$\mathbf{P}^{\times n} = \times_{\mathcal{F}_n}x$$

where $\mathcal{F}_n$ stands for the family of problems resulting of the non injective application of the initial segment of length n of natural numbers on the set $\{\mathbf{P}\}$.

II.3.4. Closures of problems.

II.3.4.1. **Definition**. $\mathbf{P}^{**} = \sum_{n=0}^{\infty} \mathbf{P}^{*n}$.

II.3.4.2. **Definition.** $P^{\times *} = \sum_{n=0}^{\infty} P^{\times n}$.

II.4. Relations Between Problems.

II.4.1. Relaxation.

II.4.1.1. **Definition.** Given problems **P** and **Q**, we use **P** ⅃ **Q** to denote **P** is a *relaxation* of **Q** in the sense:

$$\mathbf{P} \lrcorner \mathbf{Q} \leftrightarrow D_{\mathbf{Q}} \subseteq D_{\mathbf{P}} \wedge \mathcal{P}|_{D_{\mathbf{Q}}} \subseteq \mathcal{Q}.$$

Notice that $\mathbf{P} \lrcorner \mathbf{Q} \wedge \mathbf{P} \neq \mathbf{0} \to R_{\mathbf{P}} \cap R_{\mathbf{Q}} \neq \emptyset$. So, relaxation places very weak constraints on the result domains.

II.4.2. Additive Subproblems.

II.4.2.1. **Definition.** A problem **P** is an *additive subproblem* of a problem **Q** (denoted by $\mathbf{P} \sqsubseteq \mathbf{Q}$) iff there exist a problem **R** such that **P**+**R**=**Q**.

II.4.2.2. **Theorem.** The following statements are equivalent:

i. $\mathbf{P} \sqsubseteq \mathbf{Q}$,

ii. $\mathbf{P} + \mathbf{Q} = \mathbf{Q}$,

iii. $D_{\mathbf{P}} \subseteq D_{\mathbf{Q}} \wedge R_{\mathbf{P}} \subseteq R_{\mathbf{Q}} \wedge \mathcal{P} \subseteq \mathcal{Q}$.

II.4.2.3. **Definition.** By saying that a problem **P** is a *complete additive subproblem* of a problem **Q** (denoted by $\mathbf{P} \sqsubseteq_c \mathbf{Q}$) we means:

$$\mathbf{P} \sqsubseteq_c \mathbf{Q} \leftrightarrow \mathbf{P} \sqsubseteq \mathbf{Q} \wedge D_{\mathbf{P}} = D_{\mathbf{Q}}.$$

II.4.2.4. **Theorem.** The following statements are equivalent:

i. $\mathbf{P} \sqsubseteq_c \mathbf{Q}$,

ii. $\mathbf{P} \sqsubseteq \mathbf{Q} \wedge \mathbf{1}_{\mathbf{P}} = \mathbf{1}_{\mathbf{Q}}$.

iii. $\mathbf{P} \sqsubseteq \mathbf{Q} \wedge \mathbf{P} \lrcorner \mathbf{Q}$.

II.4.2.5. **Theorem.** $\mathbf{P} \lrcorner \mathbf{Q}$, $\mathbf{P} \sqsubseteq \mathbf{Q}$ and $\mathbf{P} \sqsubseteq_c \mathbf{R}$ are transitive.

II.5. Non-Algorithmic Operations.

II.5.1. Difference of Problems.

II.5.1.1. **Definition.** Let **P** and **Q** be problems:

$$\mathbf{R} = \mathbf{P} - \mathbf{Q} \leftrightarrow D_{\mathbf{P}-\mathbf{Q}} = D_{\mathbf{P}} - D_{\mathbf{Q}} \wedge R_{\mathbf{P}-\mathbf{Q}} = R_{\mathbf{P}} - R_{\mathbf{Q}} \wedge$$
$$\mathcal{P} - \mathcal{Q} = (\mathcal{P} - \mathcal{Q}) \cap (D_{\mathbf{P}-\mathbf{Q}} \times R_{\mathbf{P}-\mathbf{Q}}).$$

We say that **R** is the *complement* of **Q** (and vice versa) with respect **P**, which will be denoted by **R** =**Q**' when **P** is clear from the context.

II.5.2. **Theorem.**

$$\mathbf{Q} \sqsubseteq \mathbf{P} \to [\mathbf{R} = \mathbf{P} - \mathbf{Q} \leftrightarrow \mathbf{Q} + \mathbf{R} = \mathbf{P} \wedge (\forall \mathbf{S})(\mathbf{Q} + \mathbf{S} = \mathbf{P} \to \mathbf{R} \sqsubseteq \mathbf{S})].$$

In other words, when $\mathbf{Q} \Subset \mathbf{P}$, $\mathbf{P}-\mathbf{Q}$ is the smallest subproblem of $\mathbf{P}$ that added to $\mathbf{Q}$ yields $\mathbf{P}$.

II.5.2. Inverse Problem.

II.5.2.1. **Definition**. The *inverse* of a problem $\mathbf{P}=\langle D_\mathbf{P}, R_\mathbf{P}, \mathcal{P}\rangle$ is a problem $\mathbf{P}^{-1}=\langle R_\mathbf{P}, D_\mathbf{P}, \mathcal{P}^{\smile}\rangle$, where $\mathcal{P}^{\smile}$ is the *converse*[1] of $\mathcal{P}$.

II.6. ALGEBRAIC STRUCTURES OF PROBLEMS.

II.6.1. **Theorem.**

i. $\langle \mathbb{P}, \lrcorner\rangle$ is a preorder,

ii. $\langle \mathbb{P}, \Subset_c\rangle$ is a partial order.

iii. $\langle \mathbb{P}, \Subset\rangle$ is a complete lattice with $\mathbf{lub}=\infty$ and $\mathbf{glb}=\mathbf{0}$.

Let $\mathbf{P}_+$ and $\mathbf{P}_{c+}$ be respectively the sets of all additive subproblems and of all complete additive subproblems of $\mathbf{P}$, then:

II.6.2. **Theorem.**

i. $\langle \mathbf{P}_+, \Subset\rangle$ is a complete lattice with $\mathbf{lub}=\mathbf{P}$ and $\mathbf{glb}=\mathbf{0}$,

ii. $\langle \mathbf{P}_{c+}, \Subset\rangle$ is a complete lattice with $\mathbf{lub}=\mathbf{P}$ and $\mathbf{glb}=\mathbf{0}$,

iii. $\langle \mathbf{P}_{c+}, \Subset_c\rangle = \langle \mathbf{P}_{c+}, \Subset\rangle$.

II.6.3. **Theorem.** $\langle \mathbb{I}, \Subset\rangle$ is a complete Boolean algebra with $\mathbf{lub}=\mathbf{1}_{\mathbb{U}}$ and $\mathbf{glb}=\mathbf{0}$ and the complement of $\mathbf{I}$ is $\mathbf{1}_{\mathbb{U}}-\mathbf{I}$ for every $\mathbf{I}\in\mathbb{I}$.

II.6.4. **Definition**. Given a set of special problems $\mathbb{S}$, which will be called *easy problems*, by the *algebra of problems* over $\mathbb{S}$ we mean the algebra:

$$\mathcal{A}_{(\mathbb{S})} = \langle \mathbb{P}, \mathbb{0}, \mathbb{1}, \mathbb{2}, \mathbb{3}, \ldots, \{\Pi_i : i\in\mathbb{N}\}, \infty, \mathbb{S}, \{+, \circ, \times, -, \Sigma, ^{-1}, ^{\times n}, ^{\times *}, ^{**}\}\rangle,$$

where: $\mathbb{P}, \mathbb{0}, \mathbb{1}, \mathbb{2}, \mathbb{3}, \ldots, \{\Pi_i : i\in\mathbb{N}\}, \infty$, and $\{+, \circ, \times, -, \Sigma, ^{-1}, ^{\times n}, ^{\times *}, ^{**}\}$ are as previously defined.

II.7. PROPERTIES OF SOME SPECIAL PROBLEMS.

II.7.1. **Notation**. Given a problem $\mathbf{P}$ we will denote by $\overline{\mathbf{P}} = \langle \mathcal{D}om(\mathcal{P}), \mathcal{R}an(\mathcal{P}), \mathcal{P}\rangle$.

Notice that $\overline{\mathbf{P}}$ is the maximum gentle additive subproblem of $\mathbf{P}$, i.e. $\overline{\mathbf{P}} \Subset \mathbf{P} \wedge (\forall \mathcal{S})(\mathcal{S}\in\mathbb{G} \wedge \mathcal{S}\Subset\mathbf{P} \rightarrow \mathcal{S}\Subset \overline{\mathbf{P}})$.

[1] Here, the converse of a relation $\mathfrak{R}$ is, as in [Tar41] and [McK40], the relation $\mathfrak{R}^{\smile}$ such that for all x and y, $x\mathfrak{R}^{\smile}y$ iff $y\mathfrak{R}x$.

II.7.2. **Theorem.**

i. $1_{\overline{P}} \sqsubseteq 1_P$,

ii. $1^{\overline{P}} \sqsubseteq 1^P$,

iii. $1_{\overline{P}} = 1_P \leftrightarrow P$ is viable,

iv. $1^{\overline{P}} = 1^P \leftrightarrow P$ is surjective,

v. $1_{\overline{P}} \circ P \circ 1^{\overline{P}} = P \leftrightarrow P$ is gentle.

II.7.3. **Remark.** $D_P \cap D_Q = \emptyset \leftrightarrow 1_P \circ 1_Q = O_{P\circ Q}$,

Identities can be used to denote restriction of a problem to a subset of $\mathbb{U}$. Thus, $P|_{D_Q} = 1_Q \circ P$ and similarly for 1^Q. This gives a convenient way to express decisions in deriving programs. Notice that in our unified framework, instead of using subsets of $\mathbb{U}$, we work with the corresponding identities, which is very advantageous.

II.7.4. **Theorem.**

i. P is viable $\leftrightarrow 1_P \sqsubseteq P \circ P^{-1}$,

ii. P is deterministic $\leftrightarrow P^{-1} \circ P \sqsubseteq 1^P$,

iii. P is injective $\leftrightarrow P \circ P^{-1} \sqsubseteq 1_P$,

iv. P is surjective $\leftrightarrow 1^P \sqsubseteq P^{-1} \circ P$

v. P is surjective $\leftrightarrow P^{-1}$ is viable,

vi. P is deterministic $\wedge$ P is surjective $\leftrightarrow P^{-1} \circ P = 1^P$,

vii. P is injective $\wedge$ P is viable $\leftrightarrow P \circ P^{-1} = 1_P$,

viii. $P \lrcorner Q \leftrightarrow 1_Q \circ P \sqsubseteq_C 1_P \circ Q \circ 1^P$.

II.8. MONOTONICITY.

As usual, given relation $<$ over a set $\mathbb{G}$, we say that an operation $\ominus$ is *left monotonic* on $\langle \mathbb{G}, <\rangle$ iff for every A, B and C in $\mathbb{G}$, $A<B \rightarrow A\ominus C < B\ominus C$ holds. Analogously, we say that $\ominus$ is *right monotonic* iff $A<B \rightarrow C\ominus A < C\ominus B$ holds. Finally, we say that $\ominus$ is *monotonic* iff it is both left and right monotonic.

II.8.1. **Theorem.**

$+$ is monotonic on $\langle \mathbb{P}, \sqsubseteq_C\rangle$,

$\times$ is monotonic on $\langle \mathbb{P}, \sqsubseteq_C\rangle$,

$\circ$ is right monotonic on $\langle \mathbb{P}, \sqsubseteq_C\rangle$.

The following theorem states conditions for the left monotonicity of the product on $\langle \mathbb{P}, \sqsubseteq\rangle$.

II.8.2. **Theorem.** Given problems $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{R}$ such that $\mathbf{P}\underline{\in}_c\mathbf{Q}$:

i. sufficient conditions:

i.1 $\overline{1^{\overline{1_{\overline{P\circ Q}}\circledast P}}}\ \underline{\in}\, 1_{\overline{Q}} \rightarrow P\circledast R\ \underline{\in}_c\ Q\circledast R$,

i.2 $1^{P}\,\underline{\in}\, 1_{Q} \rightarrow P\circledast R\ \underline{\in}_c\ Q\circledast R$.

ii. necessary and sufficient condition:

$$P\circledast R\underline{\in}_c Q\circledast R \leftrightarrow \overline{1^{\overline{1_{\overline{P\circ Q}}\circledast P}}\circledast 1_{\overline{Q}}} = \overline{1^{\overline{1_{\overline{P\circ Q}}\circledast P}}}$$

The following three theorems state conditions for the monotonicity of the operations $+$, $\times$ and $\circledast$ on $\langle\mathbb{P}, \lrcorner\rangle$.

II.8.3. **Theorem.** Given problems $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{S}$ such that $\mathbf{P}\lrcorner\mathbf{Q}$ and $\mathbf{R}\lrcorner\mathbf{S}$:

i. $P+R \lrcorner Q+S \leftrightarrow 1_Q\circledast R\,\underline{\in}\, Q+1_Q\circledast S \wedge 1_S\circledast P\,\underline{\in}\, S+1_S\circledast Q$,

ii. $1_Q\circledast 1_S = \underset{Q\circledast S}{0} \rightarrow (P+R \lrcorner Q+S \leftrightarrow 1_Q\circledast R\,\underline{\in}\, Q \wedge 1_S\circledast P\,\underline{\in}\, S)$.

II.8.4. **Theorem.** Given problems $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$ such that $\mathbf{P}\lrcorner\mathbf{Q}$:

i. left monotonicity,

$$\overline{1^{\overline{1_{\overline{Q\circ R}}\circledast Q}}\circledast 1_{\overline{R}}} = \overline{1^{\overline{1_{\overline{Q\circ R}}\circledast Q}}} \rightarrow P\circledast R \lrcorner Q\circledast R,$$

ii. right monotonicity,

$$\overline{1^{\overline{1_{\overline{R\circ Q}}\circledast R}}\circledast 1_{\overline{Q}}} = \overline{1^{\overline{1_{\overline{R\circ Q}}\circledast R}}} \rightarrow R\circledast P \lrcorner R\circledast Q.$$

II.8.5. **Theorem.** Given the problems $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{S}$:

$$P\lrcorner Q \wedge R\lrcorner S \rightarrow P\times R \lrcorner Q\times S \wedge R\times P \lrcorner S\times Q.$$

III. REDUCTION, DECOMPOSITION AND SOLUTIONS.

III.1. REDUCTION.

Consider the algebra $\mathcal{A}_{\mathbb{P}}$ of definition II.6.4. We will now consider a language $\mathcal{L}_{\mathbb{P}}$ with symbols for the constants and operations of $\mathcal{A}_{\mathbb{P}}$, in addition to a set $\mathbb{V}$ of variables. Let us denote by $\mathcal{T}_{\mathbb{P}}$ the set of terms of $\mathcal{L}_{\mathbb{P}}$ and by $\mathfrak{T}_{\mathbb{P}}$ the corresponding algebra of terms [E+M87; GTW78]. In a strict fashion we should distinguish between symbols for the constants and operations, on the one hand, and their interpretations on the other hand. Nevertheless, in the sequel for the sake of simplicity, we will not always stress this distinction.

III.1.1. **Definition**. We shall call a term T of $\mathcal{T}_{\circledS}$:

i. *ground* iff it has no occurrences of variables;

ii. *algorithmic* iff all its operation symbols correspond to algorithmic operations: $+, \bullet, \times, ^{\times n}, ^{\times *}, ^{**}$;

iii. *target* iff it is ground and algorithmic.

We will refer to $\mathcal{L}_{\circledS}$ as the *global language*. We shall also consider the *ground language* as consisting of the set of ground terms. Similarly for *algorithmic* and *target languages*.

III.1.2. **Definition**. A *reduction* in $\mathcal{L}_{\circledS}$ is a pair ⟨T, T'⟩, where T and T' belongs to $\mathcal{T}_{\circledS}$.

As usual, a *valuation* is a function $s : \mathbb{H} \to \mathbb{P}$ which extends to a unique homomorphism $\bar{s} : \mathcal{T}_{\circledS} \to \mathcal{A}_{\circledS}$, assigning a problem as value to each term.

III.1.3. **Definition**. We will say that a valuation s *satisfies* the reduction ⟨T, T'⟩ iff the value of T' under s is a relaxation of the value of T under s; formally:

$$s \models \langle T, T' \rangle \leftrightarrow \bar{s}(T') \lrcorner\, \bar{s}(T).$$

III.1.4. **Definition**. We say that a reduction ⟨T, T'⟩ *is correct with respect to* a valuation s (denoted by $T' \sqsubseteq_s T$) iff s satisfies ⟨T, T'⟩ and $\bar{s}(T')$ is viable.

Notice that $\not\subseteq_c$ and = (the equality between problems) are special cases of relaxation, but they deserve attention because they have better monotonicity properties. The decision of restricting the correctness relation to one of them is part of the particular software construction strategy being used.

III.2. Decomposition.

III.2.1. **Definition**. A *decomposition schema* is a reduction of the form ⟨H, T⟩, where $H \in \mathbb{H}$.

A more general version of decomposition schema would also include a set of reductions expressing constraints on the symbols appearing in T.

III.2.2. **Definition**. We say that a decomposition schema ⟨H, T⟩ is *recursive* iff H occurs in T.

By a *decomposition schema* for a given problem **P** we mean a decomposition schema ⟨H, T⟩ in which all the occurrences of H are evaluated to **P**. We shall write this informally as $\mathbf{P} \approx T$.

III.2.3. **Definition**. Given a problem $\mathbf{P}$, a *decomposition* for $\mathbf{P}$ is a pair consisting of a decomposition schema $\langle \kappa, \mathrm{T} \rangle$ together with a valuation s such that $s(\kappa) = \mathbf{P}$ and $\mathrm{T} \sqsubseteq_s \kappa$.

III.3. SOLUTIONS.

III.3.1. **Definition**. Given a problem $\mathbf{P}$, a *solution for* $\mathbf{P}$ is a decomposition $\langle\langle \kappa, \mathrm{T}\rangle, s\rangle$ for $\mathbf{P}$, such that T is a target term.

One might wonder why we require a solution to be a decomposition of $\mathbf{P}$ into a target term. The assumption underlying this requirement is that the target machine can interpret the set $\mathbb{E}$ of problems (actually symbols for problems) and the set of operations symbols $\{+, \bullet, \times, ^{\times n}, ^{\times *}, ^{\bullet *}\}$. Since the set $\mathbb{E}$ depends on the target machine we are referring to, the property of being a target term is relative to a particular target machine.

As mentioned, $\mathbb{E}$ is the set of symbols for special constant problems, called *easy problems*. A formal definition of the concept of easy problem requires a theoretical framework beyond the scope of this paper. So, we will give an informal characterization and refer the interested reader to [H+V89a; b].

We regard a program as a term that should be interpreted by a *target machine* (a computer plus some basic software). Then, it is clear that such a term should be algorithmic. It is obvious that we cannot pretend that a target machine can interpret every problem belonging to $\mathbb{P}$. In fact, a general-purpose target machine will interpret a finite, and usually not too large, set of constant symbols over problems of $\mathbb{P}$. This is the set that we have denoted by $\mathbb{E}$.

III.4. Determinism and Non-Determinism.

As discussed in [H+V89a; b], our standpoint is that a target machine $\mathbb{H}$ together with a program $\mathrm{T}_{\mathbb{H}}$ (i.e. a target -with respect to $\mathbb{H}$- algorithmic term) such that $\mathrm{T}_{\mathbb{H}} \sqsubseteq \mathbf{P}$, for $\mathbf{P} \in \mathbb{V}$, realizes a virtual machine $\mathfrak{m}_{\mathrm{T}_{\mathbb{H}}, \mathbb{H}}$ that behaves as an engineering model of the problem denoted by $\mathbf{P}$.

Informally, we say that a virtual machine $\mathfrak{m}_{\mathrm{T}_{\mathbb{H}}, \mathbb{H}}$ is an engineering model of a problem $\mathbf{P}$ iff when we apply machine $\mathfrak{m}_{\mathrm{T}_{\mathbb{H}}, \mathbb{H}}$ to a data belonging to the data domain of $\mathbf{P}$, after certain time the machine halts and produces a result connected to this data by the condition of $\mathbf{P}$. In other words, $\mathfrak{m}_{\mathrm{T}_{\mathbb{H}}, \mathbb{H}}$ will compute the relation Q that is the condition of a problem $\mathbf{Q}$ obtained by interpreting T on $\mathbb{H}$ and $\mathbf{Q}$ is a relaxation of $\mathbf{P}$.

Thus, if $\mathfrak{m}_{\mathrm{T}_{\mathbb{H}}, \mathbb{H}}$ is a deterministic machine then the problem $\mathbf{Q}$ resulting of the interpretation of T should be deterministic (i.e. the elements of $\mathbb{E}$ should be symbols denoting deterministic problems and the operation symbols should be interpreted in a deterministic fashion). In this case the relation Q computed by $\mathbf{m}_{\mathbf{T_H}, \mathbf{H}}$ is an extension of a Skolem function of the condition P of $\mathbf{P}$.

On the other hand, what will be the situation if $\mathbf{m}_{\mathbf{T_H}, \mathbf{H}}$ is a non-deterministic machine? We can assume, without loss of generality, that the machine computes a non-empty set Ω_Q of extensions of Skolem functions of the above condition Q.

In general we can say that the machine $\mathbf{m}_{\mathbf{T_H}, \mathbf{H}}$ computes a set Ω_Q as above. Thus, Ω_Q can be regarded as a solution for problem $\mathbf{P}$ (provided that $\mathbf{Q} \lrcorner \mathbf{P}$), but not in the same sense as in definition III.2.1. The concept of solution introduced by definition III.2.1. is an intensional one related to the concept of *solving*, on the other hand, the idea of Ω_Q being a solution is more an extensional one, related to the the restriction of the choice involved in accepting condition P as a solution for $\mathbf{P}$.

IV. IV REVISITING SOME MOTIVATIONS.

At this point we can do some comparison between our initial motivations and our theoretical framework. We use relational problems as theoretical counterparts of *actual problems* to be solved (usually known as applications), as the objects denoted by specifications, as well as the theoretical counterpart of the basic modules built into the target machine.

As was discussed in [H+V89a;b], we assume that every actual problem, well or ill defined, has a theoretical counterpart. But, we do not have direct access to this theoretical counterpart; all we can hope to achieve is a formal description whose denotation is a relaxation of it.

Relaxation also reflects correctness relations between formal objects, be they programs, specifications, or theoretical counterparts of actual problems.

On the other hand, being viable is a theoretical concept to capture the fact that an application covers its domain, i.e. that it is well posed. Viability, being a theoretical concept, can be established or refuted by theoretical arguments, which cannot be done with the actual problem.

Relaxation is a relation between theoretical objects and viability is a property of such objects. Thus, relaxation and viability preservation between denotations of formal objects reflect the analytic aspects of the software development process. On the other hand, the need to resort to the extension of the actual problem in validating relaxation and in backtracking due to a failure of viability are sources of synthetic aspects of this process.

V. TOWARDS THE PROGRAMMING CALCULUS.

V.1 SOME USEFUL PROPERTIES.

Some conventions will be convenient in order to simplify our notation.

When we write 2, 3, etc., for 2_P, 3_P, etc., should be understood that problem **P** implicitly referred is any of the problems composing the generalized direct product right multiplying 2, 3, etc. Otherwise the data domain of 2, 3, etc, should be the results domain of the problem left multiplying 2, 3, etc.

When we write Π_i, it should be understood that we are referring to the problem of the class Π_i whose data domain is the result domain of the problem left multiplying Π_i.

V.1.1.1. **Theorem.**

i. For $*$ in $\{+, \times\}$, 1_P and 1^P are homomorphisms, i.e.

$$1_{P*Q} = 1_P * 1_Q \text{ and } 1^{P*Q} = 1^P * 1^Q;$$

ii. $1_{\overline{P \circ Q}} \sqsubseteq 1_{\overline{P}}$,

iii. $1^{\overline{P \circ Q}} \sqsubseteq 1^{\overline{Q}}$,

iv. $1_{\overline{P \circ Q}} = 1_{\overline{\overline{P} \circ 1_{\overline{Q}}}}$,

v. $1^{\overline{P \circ Q}} = 1^{\overline{1^{\overline{P}} \circ Q}}$,

vi. $1_{\overline{P}} = 1^{\overline{P}^{-1}}$,

vii. $1^{\overline{P}} = 1_{\overline{P}}-1$,

viii. $1_{\overline{P}} \sqsubseteq \overline{\overline{P} \circ \overline{P}^{-1}}$,

ix. $1^{\overline{P}} \sqsubseteq \overline{\overline{P}^{-1} \circ \overline{P}}$.

At this point we will introduce some miscellaneous properties derived from those of the special problems, the direct product and the product.

V.1.1.2. **Remark.**

i. For any problem **P** is:

$(2_P \times 2_P) \circ 2 \circ (\Pi_1^{\times 2} \times \Pi_2^{\times 2}) = 2_{P \times P}$, that can be written:

$(2 \times 2) \circ 2 \circ (\Pi_1^{\times 2} \times \Pi_2^{\times 2}) = 2$,

ii. for $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{S}$ viable problems is:

$$[(P \times Q) \times (R \times S)] \circ 2 \circ (\Pi_1^{\times 2} \times \Pi_2^{\times 2}) =$$

$$2 \circ (\Pi_1^{\times 2} \times \Pi_2^{\times 2}) \circ [(P \times R) \times (Q \times S)],$$

iii. for any problems $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{S}$ is:

$$2_{(P\times Q)\times(R\times S)} \circ (\Pi_1^{\circ 2} \times \Pi_2^{\circ 2}) \circ 2 \circ (\Pi_1^{\circ 2} \times \Pi_2^{\circ 2}) =$$

$$(1_P \times 1_Q) \times (1_R \times 1_S),$$

iv. for any viable problems $\mathbf{P}$ and $\mathbf{Q}$ is:

$$2 \circ (1_P \times 1_Q) = 1_P \circ 1_Q \circ 2 \circ (\tfrac{1}{Q} \times \tfrac{1}{Q}) = 1_P \circ 1_Q \circ 2,$$

v. for any viable problems $\mathbf{P}$ and $\mathbf{Q}$ is:

$$(P \times Q) \circ \Pi_2 = (P \times 1_Q) \circ \Pi_2 \circ Q = \Pi_2 \circ Q,$$

vi. for any viable problems $\mathbf{P}$ and $\mathbf{Q}$ is:

$$(P \times Q) \circ \Pi_1 = (1_P \times Q) \circ \Pi_1 \circ P = \Pi_1 \circ P,$$

vii. for any problem $\mathbf{P}$ is:

$2_P \circ 2 = 2_P \circ (2 \times 2)$, that can be written:

$2 \circ 2 = 2 \circ (2 \times 2)$,

viii. if $\mathbf{P}$ is deterministic $\wedge$ $\mathbf{P}$ is surjective, in other words if

$P^{-1} \circ P = 1^P$, is:

$P \circ 2 = 2 \circ (P \times P)$, in general $P \circ n = n \circ P^{\times n}$

ix. for any problem $\mathbf{R}$ is:

$R \circ 2 \circ \Pi_1 = R \circ 2 \circ \Pi_2 = R$, that can be written:

$2 \circ \Pi_1 = 2 \circ \Pi_2 = 1$.

In general for every $0 \leq i \leq n$, is:

$n \circ \Pi_i = 1$,

x. for any problem $\mathbf{R}$ is:

$R \circ 3 \circ (\Pi_1 \times \Pi_2) = R \circ 3 \circ (\Pi_3 \times \Pi_2) = 2$, that can be written:

$3 \circ (\Pi_1 \times \Pi_2) = 3 \circ (\Pi_3 \times \Pi_2) = 2$. In general for every $0 \leq m \leq n$ and every $0 \leq i_1, \ldots, i_m \leq n$, is:

$n \circ (\Pi_{i_1} \times \ldots \times \Pi_{i_n}) = m$.

V.2 DECOMPOSITION SCHEMATA.

In order to illustrate some of our ideas on problem solving and algorithm development, we shall examine some versions of divide-and-conquer. The central idea of divide-and-conquer is to divide the problem P into subproblems. One of them, denoted E, is easy, because P is easy on its data domain. The remaining one consists of data that are going to be split (via a problem S) into problems, whose results will be merged (via a problem M) into a final result.

We can express an n-ary divide-and-conquer strategy by means of the recursive decomposition schema

$$\langle \mathbb{H}, E + S \circ \mathbb{H}^{\times n} \circ M \rangle.$$

Here, $\mathbb{H}$ stands for a given problem P to be solved by divide-and-conquer. So, we write this as

$$P \approx E + S \circ P^{\times n} \circ M$$

to indicate that we still have to find problems E, S and M.

A more general expression for an n-ary divide-and-conquer schema is

$$\langle \mathbb{H}, E + S \circ (T_1 \times \ldots \times T_n) \circ M \rangle,$$

where each term T_i is $\mathbb{H}$ or a constant in $\mathbb{D}$. This captures the idea that each data is split into n data to be handled independently of each other.

A very simple case of this strategy is the *unary* divide-and-conquer decomposition schema of the form:

$$P \approx E + S \circ P \circ M$$

We wish to instantiate this schema, by an appropriate choice s for E, S and M, to a decomposition for P, i.e. $E + S \circ P \circ M \sqsubseteq_s P$. This means that the extended valuation $\bar{s}(E + S \circ P \circ M)$ is a viable relaxation of P. Leaving $\bar{s}$ implicit, a strong but useful sufficient condition for this is:

$$E + S \circ P \circ M \underline{\Subset}_c P \text{ and}$$

$$S \underline{\Subset} \mathbb{W}, \text{ for some } \mathbb{W} \text{ with a well-founded condition.}$$

V.3 STRATEGIES.

V.3.1 Generalization.

With this strategy we want to capture the idea of reducing a problem to another one, as used in mathematics, for instance. By a *generalization* for a problem P we mean any problem Q occurring in a decomposition schema of the form $P \approx T \circ Q \circ T'$.

V.3.2 Trivialization.

Assume that we have a problem P that is not easy but contains an easy viable additive subproblem $Q \underline{\Subset} P$. In other words, the data domain of Q contains only data for which P is easy. Therefore, we can apply the decomposition on identities:

$$\mathbf{1}_P = \mathbf{1}_Q + (\mathbf{1}_P - \mathbf{1}_Q)$$

Clearly, the effect of this decomposition for $\mathbf{1}_P$ is to obtain a subdomain $\mathbf{1}_Q$ of the *easy data* of P and a subdomain $\mathbf{1}_P - \mathbf{1}_Q$ of remaining data of P.

Now, we can use the preceding domain decomposition to induce a decomposition for P itself, so, we can write:

$$P = \mathbf{1}_P \circ P = (\mathbf{1}_Q + (\mathbf{1}_P - \mathbf{1}_Q)) \circ P = \mathbf{1}_Q \circ P + (\mathbf{1}_P - \mathbf{1}_Q) \circ P$$

where, $\mathbf{1}_Q \circ P$ is, obviously, an easy subproblem Q and $(\mathbf{1}_P - \mathbf{1}_Q) \circ P$ is the remaining subproblem Q'

The idea is that we should annihilate Q' by converting it to zero by means of suitable manipulations. If we succeed in doing this, we are using, instead of P, a restriction of P to its easy data. In other words we *trivialize* P without making P an easy problem.

V.4 A CASE STUDY.

As an illustration of the use of our programming calculus, we will now outline the derivation of a program for the *Palindrome* problem. This problem can be informally stated as "decide whether a given sequence is a palindrome, i.e. whether it reads the same forward and backward".

H. Partsch [Par88] used a CJP-like [Par86] method to derive a program for this problem by means of extensive use of fold-unfold techniques. Our goal here is to illustrate our algebraic programming calculus, instead of presenting a original derivation. So, we will pick some of the ideas in Partsch's derivation, without following its details literally. Instead, we shall use our own strategies, as in the case of deriving decompositions from domain properties rather than imposing a schema a priori.

The domains of our problem are:

L_t = set of lists of elements from a set t, and Bool = {T, F}.

Its condition consists of:

$\langle x, y\rangle \in L_t \times$ Bool such that $y{=}T \leftrightarrow x = x^{\sim}$;

where $x^{\sim}$ denotes the reversal of x.

Thus our palindrome problem is:

$P{=}\langle L_t, \text{Bool}, \{\langle x, y\rangle \in L_t \times \text{Bool} : y{=}T \leftrightarrow x = x^{\sim}\}\rangle$.

Since our problem involves lists and Booleans, some problems (corresponding to primitive operations) will be considered easy. In order to introduce them, let us first fix some notation:

head (x) denotes the head of x,

tail (x) denotes the tail of x,

last (x) denotes the last element of x,

initial (x) denotes the result of extracting the last element of list x, *cons* (e, x) denotes the result of inserting element e into the beginning of list x,

append (x, e) denotes the result of inserting element e into the end of list x,

$|x|$ denotes the length of list x.

We can now introduce our easy problems. Some of them correspond to simple operations on lists:

$\mathbf{H} = \langle L_t, t, \{\langle x, y\rangle \in L_t \times t : |x|>0 \wedge y = \mathit{head}(x)\}\rangle$,

$\mathbf{T} = \langle L_t, L_t, \{\langle x, y\rangle \in L_t \times L_t : |x|>0 \wedge y = \mathit{tail}(x)\}\rangle$,

$\mathbf{L} = \langle L_t, t, \{\langle x, y\rangle \in L_t \times t : |x|>0 \wedge y = \mathit{last}(x)\}\rangle$,

$\mathbf{I} = \langle L_t, L_t, \{\langle x, y\rangle \in L_t \times L_t : |x|>0 \wedge y = \mathit{initial}(x)\}\rangle$,

$\mathbf{C} = \langle t \times L_t, L_t, \{\langle x, y\rangle \in (t \times L_t) \times L_t : y = \mathit{cons}(\pi_1(x), \pi_2(x))\}\rangle$,

$\mathbf{A} = \langle L_t \times t, L_t, \{\langle x, y\rangle \in (t \times L_t) \times L_t : y = \mathit{append}(\pi_1(x), \pi_2(x))\}\rangle$,

$\mathbf{M} = \langle L_t, L_t, \{\langle x, y\rangle \in L_t \times L_t : |x|>1 \wedge y = \mathit{initial}(\mathit{tail}(x))\}\rangle$.

We also have problems corresponding to simple restrictions of L_t according to length:

$\mathbf{1}_{=0} = \langle L_t, L_t, \{\langle x, y\rangle \in L_t \times L_t : y = x \wedge |x|=0\}\rangle$,

$\mathbf{1}_{\neq 0} = \langle L_t, L_t, \{\langle x, y\rangle \in L_t \times L_t : y = x \wedge |x|\neq 0\}\rangle$,

$\mathbf{1}_{\leq 1} = \langle L_t, L_t, \{\langle x, y\rangle \in L_t \times L_t : y = x \wedge |x|\leq 1\}\rangle$,

$\mathbf{1}_{>1} = \langle L_t, L_t, \{\langle x, y\rangle \in L_t \times L_t : y = x \wedge |x|>1\}\rangle$.

In addition, we have problems corresponding to some simple Boolean operations:

$\mathbf{TRUE} = \langle \mathbb{U}, \mathrm{Bool}, \{\langle x, y\rangle \in \mathbb{U} \times \mathrm{Bool} : y=\mathrm{T}\}\rangle$,

$\mathbf{FALSE} = \langle \mathbb{U}, \mathrm{Bool}, \{\langle x, y\rangle \in \mathbb{U} \times \mathrm{Bool} : y=\mathrm{F}\}\rangle$,

$\mathbf{AND} = \langle \mathrm{Bool} \times \mathrm{Bool}, \mathrm{Bool}, \{\langle x, y\rangle \in (\mathrm{Bool} \times \mathrm{Bool}) \times \mathrm{Bool} : y=\pi_1(x) \wedge \pi_2(x)\}\rangle$.

Finally, we have as an easy problem the test for equality of elements of t:

$\mathbf{E} = \langle t \times t, \mathrm{Bool}, \{\langle x, y\rangle \in (t \times t) \times \mathrm{Bool} : y=\mathrm{T} \leftrightarrow (\pi_1(x) = \pi_2(x))\}\rangle$,

Some properties of the easy problems on lists can be stated in our notation as follows:

i. $\mathbf{C} \circ \mathbf{H} = \mathbf{1}_{\mathbf{C}} \circ \Pi_1$ (which is how we state *head* (*cons* (e, x))=e),

ii. $\mathbf{C} \circ \mathbf{T} = \mathbf{1}_{\mathbf{C}} \circ \Pi_2$ (which is how we state *tail* (*cons* (e, x))=x),

iii. $\mathbf{Y} \circ \mathbf{H} = \mathbf{1}_{\mathbf{Y}} \circ \Pi_1$ where $\mathbf{Y} = [\mathbf{3} \circ (\mathbf{H} \times \mathbf{M} \times \mathbf{L})]^{-1}$

iv. $\mathbf{A} \circ \mathbf{L} = \mathbf{1}_{\mathbf{A}} \circ \Pi_2$

v. $Y \circ T = 2_Y \circ (\Pi_2 \times \Pi_3) \circ A$

vi. $2_p \circ (H \times T) = C^{-1} \wedge 2_p \circ (1 \times L) = A^{-1}$

vii. $1_{>1} = 1_{>1} \circ Y^{-1} \circ Y$

The very definition of the palindrome problem suggests restating it in terms of two auxiliary problems:

the problem of reversing lists, which we can define as:

$R = \langle L_t, L_t, \{\langle x, y\rangle \in L_t \times L_t : y = x^{\sim}\}\rangle$,

and the problem of equality on lists:

$E = \langle L_t \times L_t, \mathrm{Bool}, \{\langle x, y\rangle \in (L_t \times L_t) \times \mathrm{Bool} : y = \tau \leftrightarrow (\pi_1(x) = \pi_2(x))\}\rangle$,

We can now restate P by means of these auxiliary problems, which yields the decomposition schema:

$$\boxed{P = 2_p \circ (1 \times R) \circ E}$$

Notice that this equation compactly expresses the idea: *to check whether a sequence is a palindrome, reverse a copy of it and compare it to the original one*. It can be regarded as an algorithm (perhaps not a very efficient one) for P, since the right-hand side involves only algorithmic operations. But, problems R and E do not belong to the set of easy problems, which defines our *list manipulation language*; thus the right-hand side is not directly executable on our target machine. So, we must derive expressions for R and E in terms of easy problems.

Some useful properties of problem R are:

α. $1_{>1} \circ R = R \circ 1_{>1}$;

β. $R \circ H = L$; $R \circ M = M \circ R$; $R \circ L = H$;

We now derive a decomposition for R.

We start by separating R into an easy subproblem and its complement, as explained in IV.2.3.2:

$R = 1_R \circ R = (1_{\leq 1} + 1_{>1}) \circ R = 1_{\leq 1} \circ R + 1_{>1} \circ R$

On the one hand $1_{\leq 1} \circ R = 1_{\leq 1} \circ 1_R = 1_{\leq 1}$

On the other hand:

$1_{>1} \circ R = R \circ 1_{>1}$ (by α)

$= R \circ 1_{>1} \circ Y^{-1} \circ Y$ (by vii)

$= 1_{>1} \circ R \circ Y^{-1} \circ Y$ (by α)

$= 1_{>1} \circ R \circ 3 \circ (H \times M \times L) \circ Y$ (since $[Q^{-1}]^{-1} = Q$ for any Q)

$= 1_{>1} \circ 3 \circ (R \times R \times R) \circ (H \times M \times L) \circ Y$ (by remark V.1.1.2)

$= 1_{>1} \circ 3 \circ (R \circ H \times R \circ M \times R \circ L) \circ Y$ (by theorem II.3.3.3)

$= 1_{>1} \circ 3 \circ (L \times M \circ R \times H) \circ Y$ (by β)

Therefore,

$$\boxed{R = 1_{\leq 1} + 1_{>1} \circ 3_p \circ (L \times (M \circ R) \times H) \circ Y}$$

Notice that this decomposition schema is an instance of the divide-and-conquer schema $R = \mathrm{Easy} + \mathrm{Split} \circ (1 \times \ R \times 1) \circ \mathrm{Merge}$, since the right-hand side can be rewritten as $1_{\leq 1} + [1_{>1} \circ 3_p \circ (L \times M \times H)] \circ (1 \times R \times 1) \circ Y$.

By means of similar manipulations and applying Remark V.1.1.2 and Theorem II.3.3.3, we obtain the following decomposition for E:

$$\boxed{\begin{array}{l} E = S + (1_{\neq 0} \times 1_{\neq 0}) \circ 2 \circ (H^{\times 2} \times T^{\times 2}) \circ (E \times E) \circ \textsc{And} \\ \text{where:} \\ S = (1_{=0} \times 1_{=0}) \circ \textsc{True} + (1_{=0} \times 1_{\neq 0}) \circ \textsc{False} + (1_{\neq 0} \times 1_{=0}) \circ \textsc{False} \end{array}}$$

At this point, we can collect the results obtained so far in the form of the following equation system, where we have already applied our conventions for forks and identities:

$$\boxed{\begin{array}{ll} (1) & P = 2 \circ (1 \times R) \circ E \\ (2) & R = 1_{\leq 1} + 1_{>1} \circ 3 \circ (L \times (M \circ R) \times H) \circ Y \\ (3) & E = S + (1_{\neq 0} \times 1_{\neq 0}) \circ 2 \circ (H^{\times 2} \times T^{\times 2}) \circ (E \times E) \circ \textsc{And} \\ (4) & S = (1_{=0} \times 1_{=0}) \circ \textsc{True} + (1_{=0} \times 1_{\neq 0}) \circ \textsc{False} + (1_{\neq 0} \times 1_{=0}) \circ \textsc{False} \\ (5) & Y = [3 \circ (H \times M \times L)]^{-1} \end{array}}$$

From this system we can derive an explicit solution for P by applying fold-unfold techniques together with our algebraic laws.

By unfolding R in equation (1) according to (2), we obtain:

$$\begin{array}{lr} P = 1_{\leq 1} \circ 2 \circ E + & (a) \\ \quad 1_{>1} \circ 2 \circ (1 \times 3 \circ (L \times (M \circ R) \times H) \circ Y) \circ E & (b) \end{array}$$

By some simple manipulations on the first term of the right-hand side we have:

$$(a) = 1_{\leq 1} \circ \textsc{True}$$

Now, by unfolding E in the second term of the right-hand side according to (3), we obtain by some manipulations:

$$\begin{array}{l} (b) = 1_{>1} \circ 2 \circ (1 \times 3 \circ (L \times (M \circ R) \times H) \circ Y) \circ 2 \circ \\ \qquad\qquad\qquad\qquad (H^{\times 2} \times T^{\times 2}) \circ (E \times E) \circ \textsc{And} \end{array}$$

By applying Remark V.1.1.2 , we obtain:

$$\begin{array}{l} (b) = 1_{>1} \circ 2 \circ \\ \qquad \{[2 \circ (H \times L) \circ E] \times \\ \qquad\qquad\qquad [2 \circ (T \times (2 \circ (M \circ R \times H) \circ R)) \circ E]\} \\ \qquad\quad \circ \textsc{And} \end{array}$$

Let us denote by (d) the expression within the second pair of square brackets. By operating on it we have:

(6) $(d) = 2 \circ (2 \times 2) \circ ((M \times L) \times (M \circ R \times H)) \circ 2 \circ (\Pi_1^{\times 2} \times \Pi_2^{\times 2}) \circ 2$
$\circ (\Pi_1^{\times 2} \times \Pi_2^{\times 2}) \circ A^{\times 2} \circ E$

Now, let us recall that:

if $x_1, x_2 \in t$ and $l_1, l_2 \in L_t$, then:

$x_1 = x_2 \wedge l_1 = l_2 \leftrightarrow cons(x_1, l_1) = cons(x_2, l_2)$.

This property can be stated in our algebraic style as follows:

$(E \times \mathcal{E}) \circ \text{AND} = 2 \circ (\Pi_1^{\times 2} \times \Pi_2^{\times 2}) \circ A^{\times 2} \circ E$

By applying this property together with points (i) and (ii) of Remark V.1.1.2 to (6), we obtain:

$(d) = 2 \circ 2 \circ ((M \times M \circ R) \times (L \times H)) \circ (E \times \mathcal{E}) \circ \text{AND}$

The use of (d) in (b) yields:

$(b) = 1_{>1} \circ 2 \circ$

$\{[2 \circ (H \times L) \circ \mathcal{E}] \times$

$[2 \circ 2 \circ ((M \times M \circ R) \times (L \times H)) \circ (E \times \mathcal{E}) \circ \text{AND}]\}$

$\circ \text{AND}$

By applying AND properties, Theorem II.3.3.3 and folding on P, we obtain:

$(b) = 1_{>1} \circ 2 \circ [(2 \circ (H \times L)) \times M] \circ (\mathcal{E} \times P) \circ \text{AND}$

Finally:

$$\boxed{P = 1_{\leq 1} \circ \text{TRUE} + 1_{>1} \circ 2 \circ [(2 \circ (H \times L)) \times M] \circ (\mathcal{E} \times P) \circ \text{AND}}$$

By means of some manipulations we can rewrite the preceding decomposition as an iterative one:

$P = 1_{\leq 1} \circ \text{TRUE} + 1_{>1} \circ 2 \circ (2 \circ (H \times L) \circ \mathcal{E} \times M) \circ (1_{\text{FALSE}} \times 1) \circ \Pi_1 +$

$1_{>1} \circ 2 \circ (2 \circ (H \times L) \circ \mathcal{E} \times M) \circ (1_{\text{TRUE}} \times 1) \circ \Pi_2 \circ P$

which is of the form:

$P = x + y \circ P$

that is a unary iterative divide-and-conquer schema.

VI. CONCLUSIONS.

We have given theoretical bases of a calculus for program construction. This calculus is based on an algebra of problems. In this framework we have expressed some basic programming concepts, such as correctness, decomposition, etc.

Our calculus so far is oriented towards the construction of programs from formal specification. But, we have developed it having in mind its extension to a calculus for the construction of software pieces from actual problems.In fact, exploratory research analyzing some aspects of the context of such a calculus is well under way, some preliminary results having been reported [V+H89b; H+V89a,b].

The framework presented here is just one among several possibilities, based on slight variations of our basic concepts. We have decided to work on the model-theoretical level. We have resorted to syntactical aspects in defining solution and decomposition. in section III. This was mainly fcr the sake of simplicity; we are not (yet) proposing a language.

We think that our calculus is quite helpful in deriving programs from formal specifications. It appears to be able to express,in a convenient way, some useful strategies, as illustrated by generalization, trivialization and divide-and-conquer, as well as in deriving properties by means of algebraic manipulations These ideas are illustrated in the development of a simple example . We trust that this example will give some idea of the heuristic power of the notation and the calculus.

Three important aspects of our calculus are being analyzed with the goal of extending it. First, we are collecting more evidence concerning its use in program derivation. Second, we are starting to use it to analyze and manipulate program construction methods and strategies themselves (in a "second order" fashion). Third, we continue the studies towards introducing synthetic aspects into the calculus.

REFERENCES.

[EVHV89] Elustondo, P., Veloso, P., A., S., Haeberer, A., M.,Vazquez, L.:
Program Development in the Algebraic Theory of Problems- XVIII Jornadas Argentinas de Informática e Investigación Operativa. Buenos Aires. September 1989.

[GTW78] Goguen, J., Thatcher, J., Wagner, E., Wright, J.:
An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types. Current Trends on Programming Methodology , Vol IV, Yeh, R. (Ed.), Prentice Hall, Englewood Cliffs, N. J., 1978, pp. 80-144.

[HBV87] 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; Rio de Janeiro, 1987.

[HVB89] Haeberer, A., M., Veloso P., A., S., Baum G. :
Formalización del Proceso de Desarrollo de Software. Kapeluz, Buenos Aies, 1989.

[H+V89a] Haeberer, A., M., Veloso P., A., S.:
The Requirement-Specification-Program Triangle: A Theoretical Analysis. Pont. Universidade Católica; Res. Rept. MCC 7/89; Rio de Janeiro,1989. Submitted to Formal Aspects of Computing.

[H+V89b] Haeberer, A., M., Veloso P., A., S.:
The Invitability of program testing - A theoretical analysis.Proc. of the XVII Chilean Conference on Informatics. July 1989.

[JHW86] Jähnichen, S., Ali Hassain, F. and Weber, M.:
Program Development Using a Design Calculus. GMD Forschungsstelle an der Universität Karlsruhe, 1986.

[Leh84] Lehman, M., M.:
A Further Model of Coherent Programming Process. IEEE Proceedings of Software Process Workshop, UK Feb 1984, IEEE Comp. Soc.,1984.

[LST84] Lehman, M., M., Stenning V. and Turski W., M.:
Another Look at Software Desing Methodology. ICST DoC Res. Rep., 83/13, London SW7 2BZ, Jun 1983.

[L+B85] Lehman M., M., Belady L., A. Eds.:
Program Evolution: Processes of Software Change. Academic Press, London, 1985.

[M+T84] Maibaum, T., S., E. and Turski, W., M.:

On What Exactly is Going On When Software is Developed Step by Step. Proc. 7th ICSE, Orlando, FA. March 1984. Publ. IEEE Comp. Soc., Silver Spring, MA. IEEE cat. no. 84ch2011-5, pp 525-533.

[McK40] McKinsey, J., C., C.,:
Postulates for the Calculus of Binary Relations. Journal of Symbolic Logic, V. 5, N. 3, September1940.

[Par86] Partsch, H., :
Transformational Program development in a Particular Problem Domain. Science of Computer Programming.V.7 N.21 September 1986.

[Par88] Partsch, H., :
Specification and Transformation of Programs- A Formal Approach to Software Development. Privatelly circulated.1988

[Sin85] Sintzoff, M.:
Desiderata for a Design Calculus. UCL, T3 Memo, Unité d'Informatique, Univ. Louvain, June 1985.

[Sin86] Sintzoff, M.:
Playing with a Recursive Design Calculus. UCL, T3 Memo, Unité d'Informatique, Univ. Louvain, June 1986.

[Sup61] Suppes, P.:
Axiomatic Set Theory. D. Van Nostrand Company Inc., New York, 1961.

[Tar41] Tarski, A.:
On The Calculus of Relations. Journal of Symbolic Logic, V. 6, N. 3, September1941.

[T+M87] Turski, W., M., Maibaum, T., S., E.:
The Specification of Computer Programs. Addison-Wesley, Wokingham, 1987.

[Vel84] Veloso, P., A., S.:
Outline of a mathematical theory of general problems. Philosophia Naturalis; 21(2-4), 1984, pp. 354-365.

[V+H89a] Vargas, D., C., Haeberer, A., M.
Formal Theories of Problems. Pont. Universidade Católica; Res. Rept. MCC6/89; Rio de Janeiro, 1989.

[V+H89b] Veloso, P., A., S., Haeberer, A., M.
Software Development: A Problem-theoretic Analysis and Model. 22nd Hawaii International Conference on System Science, Honolulu, January 1989.

[V+E89] Vazquez, L., A., Elustondo, P., Towards Program Construction on the Algebraic Theory of Problems. M.XVIII Jornadas Argentinas de

Informática e Investigación Operativa. Buenos Aires. September1989.