THE ROLE OF "DIVIDE" IN DIVIDE-AND-RULE ALGORITHMS

O. de Moor

RUU-CS-87-26
November 1987

Rijksuniversiteit Utrecht

Vakgroep informatica

Budapestlaan 6 3584 CD Utrecht
Corr.adres: Postbus 80.012 3508 TA Utrecht
Telefoon 030-53 1454
The Netherlands

THE ROLE OF "DIVIDE" IN DIVIDE-AND-RULE ALGORITHMS

O. de Moor

Technical Report RUU-CS-87-26
November 1987

Department of Computer Science
University of Utrecht
P.O.Box 80.012, 3508 TA Utrecht
The Netherlands

Abstract

A most promising approach to transformational program development is the manipulation of structure–preserving mappings, as advocated by L.G.L.T. Meertens and R.S. Bird. They view such functions as homomorphisms on algebraic structures. This excludes general divide–and–rule algorithms from the calculus of transformations, since homomorphisms do not allow for explicit specification of 'divide'. In this report, a general scheme for divide–and–rule is introduced. Under extremely weak conditions, two consecutive divide–and–rule algorithms may be merged into a single one. The laws on homomorphisms used by Meertens and Bird are corollaries to this general composition theorem. The formal framework developed to prove the theorem permits a concise formulation of the popular transformation technique known as "formal differentiation". A possible application is the development of algorithms on sets.

Contents

1 Introduction 2

2 Preliminaries 4
- 2.1 Logic . . . 4
- 2.2 Abstract types . . . 5
- 2.3 Indeterminacy . . . 8
- 2.4 Refinement . . . 10

3 Theory 12
- 3.1 Manageable structures . . . 12
- 3.2 Homomorphisms . . . 21
- 3.3 Indeterminate insertion . . . 30
- 3.4 General divide–and–rule: Integrates . . . 35
  - 3.4.1 Specification of the divide strategy . . . 36
  - 3.4.2 Integrates . . . 39
  - 3.4.3 Composition of integrates . . . 45

4 Formal integration 60
- 4.1 Applications . . . 62
  - 4.1.1 Integrate products . . . 62
  - 4.1.2 Set partitions . . . 64

5 Discussion 66

Bibliography 68

Index 71

Chapter 1

Introduction

In the initial stages of program development, it is good practice to specify as little as possible. The reason for this admonition is obvious: the fewer constraints a specification contains, the more susceptible it is to formal manipulation. No undue commitments should be made. Most principles of transformational programming are rooted in this Law of Parsimony. The current emphasis on functional programs may be explained by the fact that purely functional specifications[1] are not necessarily blurred by redundant details on execution order, as imperative programs are.

This paper investigates the need and possibilities to specify the divide strategy in divide–and–rule algorithms. Often the ways of splitting an argument are crucial and should be restricted. For instance, the number (#) of elements in a set is not adequately defined by:

$$\begin{aligned} \#\,\emptyset &= 0 \\ \#\,\{a\} &= 1 \\ \#\,(x \cup y) &= \#x + \#y. \end{aligned}$$

The third clause needs the constraint that x and y are disjoint. This amounts to a restriction of the divide strategy. We will identify conditions stating whether such restrictions are necessary or not.

For programs without a specific divide policy, elegant transformation laws can be formulated. This has been done in various papers by Meertens and Bird (notably [Mee86] and [Bir86b]). The approach is to regard such

[1]The terms 'program', 'algorithm', 'function' and 'expression' are used interchangeably to denote specifications.

algorithms as homomorphisms on algebraic structures. The theory might be called 'morphic programming'.

Generalizing their work, we will develop a calculus of divide–and–rule algorithms, incorporating explicit divide strategies. A theorem on the merging of two consecutive programs will be proved. An important part of morphic programming are the so–called "promotion laws", which will be presented as corollaries to this general composition theorem.

At the outset of these investigations, morphic programming was still lacking the sound mathematical basis needed to carry them through. The situation changed when Zantema presented a treatment of the theory along the lines of universal algebra [Zan87]. Certain aspects of his foundation, however, seemed incompatible with the research described in this paper, and therefore another approach is outlined in section 3.1. After discussing homomorphisms and the basic operations of morphic programming (section 3.2), we turn to functions lacking a divide strategy (section 3.3).

Next the more general form of divide–and–rule algorithms (involving explicit specification of 'divide') is introduced. The new construct is christened 'integrate', because of analogies with conventional integration. The composition theorem mentioned above is proved at this point (section 3.4).

Several authors have noted the resemblance between the development of loops and differentiation as it is known from ordinary calculus (e.g. [Sha81,Sha82,PK82,Pai86,Mee87b]). The issue is revisited, and accentuated by a suggestive notation (chapter 4). In this context some programming problems on sets will be considered.

Finally, the question is raised what might be gained from the results obtained in this paper. Though some nice little programming problems could be solved with the help of the method developed here, it does not yet seem suited for deriving more substantial algorithms. Therefore, the feasibility of extending our theory to more complex structures as well as other possible directions for future research are briefly discussed in chapter 5.

This paper does not stand by itself. Some level of familiarity with the earlier work on morphic programming is assumed. A good introduction is [Mee87a].

Chapter 2

Preliminaries

As noted in the introduction, a sound mathematical basis is needed to solve the problems that are the subject of this report. A modest attempt will be made to treat morphic programming in the framework of algebraic data types. The relation between such types and transformational programming has been extensively studied in the project CIP at the Technical University of Munich.

Some basic concepts and notations will be briefly discussed. The reader is referred to [BBB*85,BPW80,WPP*83] for technical details. Despite the adoption of the basics from CIP, we adhere to the free syntax commonly used in morphic programming (e.g. [BMW85], Meertens' proposal). To avoid confusion with other notions of 'algebra' in this report, algebraic types will be called abstract types as in [WPP*83].

2.1 Logic

In abstract types, first order predicate logic (together with equational logic) is used. No special properties of this calculus are important, except that it is sound: If something is provable, it is true. The symbol '$\vdash$' denotes the relation 'proves', and '$\models$' stands for 'models'. Thus, soundness is expressed by:

$$\Gamma \vdash \phi \Rightarrow (\mathcal{A} \models \Gamma \Rightarrow \mathcal{A} \models \phi).$$

To avoid confusion between syntax and semantics, '$\longrightarrow$' is used as the logical implication symbol, while '$\Rightarrow$' specifies this relation in the real world.

Another notion used both in predicates and on the meta–level is universal quantification. Here the distinction is made clear by a different notation for the range of the variable. In predicates, we write

$$\forall s\ x : \phi(x),$$

whereas a meta–sentence might typically read

$$\forall t \in S : p(t).$$

2.2 Abstract types

Abstract types provide a means for specifying sets of objects together with operations on these sets. The sets are denoted by sort symbols, often called *sorts* for short. The operations are written with an *operation name*, which is associated with a so–called functionality. A *functionality* indicates the domain and codomain of the operation. For example, if 'nat' is a sort denoting the natural numbers, and '+' is the name of addition, then $\mathsf{nat} \times \mathsf{nat} \rightarrow \mathsf{nat}$ is the functionality of '+'. This is written

$$(+ : \mathsf{nat} \times \mathsf{nat} \rightarrow \mathsf{nat}) \in T_{opns}.$$

An *abstract type* T is a pair $< T_{sign}, T_{laws} >$. The *signature* T_{sign} is a pair $< T_{sorts}, T_{opns} >$). T_{sorts} is a set of sort symbols, and T_{opns} is a collection of operation names with functionalities. The *language of* T, consisting of terms constructed from variables and the symbols in T_{opns}, is denoted by $\mathsf{terms}(T_{sign})$. The set $\mathsf{terms}(T_{sign})$ is *sorted* by T_{sorts}; thus, for $s \in T_{sorts}$, $\mathsf{terms}(T_{sign})_s$ stands for the expressions of sort s. The *ground expressions*, not containing any variables, will be referred to by $\mathsf{grterms}(T_{sign})$.

The *laws* (T_{laws}) are first order predicates over the language of T_{sign}. These laws characterize the possible interpretations of the symbols in the signature. An example is provided below. The phrase "**based on** INT, BOOL" specifies that integers and booleans, together with their operations, are available in this abstract type. The notion of availability may be quite subtle [WPP*83]. For the moment, details are ignored here.

Example 1

type	SEQU =		
based on	INT, BOOL		
sorts	iseq		
opns	[]	:	$\rightarrow$ iseq
	[_]	:	int $\rightarrow$ iseq
	$[\_]_{even}$	:	int $\rightarrow$ iseq
	$\_ +\!\!+ \_$	:	iseq $\times$ iseq $\rightarrow$ iseq
	$\_ \widetilde{+\!\!+} \_$	:	iseq $\times$ iseq $\rightarrow$ iseq
	hd_	:	iseq $\rightarrow$ int
	tl_	:	iseq $\rightarrow$ iseq
	lead_	:	iseq $\rightarrow$ iseq
	last_	:	iseq $\rightarrow$ int
laws	$\forall$iseq x, y, z	:	$(x +\!\!+ y) +\!\!+ z = x +\!\!+ (y +\!\!+ z)$
	$\forall$iseq x	:	$(x +\!\!+ [\,]) = x$
	$\forall$iseq x	:	$([\,] +\!\!+ x) = x$
	$\forall$iseq x, y	:	$(x \widetilde{+\!\!+} y) = (y +\!\!+ x)$
	$\forall$int n	:	$[n]_{even} =$ $even\ n \rightarrow [n]$
			$\square$ $\neg even\ n \rightarrow [\,]$
	$\forall$int n $\forall$iseq x	:	$\mathbf{hd}([n] +\!\!+ x) = n$
	$\forall$int n $\forall$iseq x	:	$\mathbf{tail}([n] +\!\!+ x) = x$
	$\forall$iseq x $\forall$int n	:	$\mathbf{last}(x +\!\!+ [n]) = n$
	$\forall$iseq x $\forall$int n	:	$\mathbf{lead}(x +\!\!+ [n]) = x$

An important class of predicates is formed by the equations that are provable from the set of laws T_{laws}. This set of equations will be called the *theory of* T,

$$\mathsf{Th}(T) := \{(t_0 = t_1) | t_0, t_1 \in \mathsf{terms}(T_{sign}), T_{laws} \vdash t_0 = t_1\}.$$

It will be convenient to have a notation for a restricted kind of equations, where not all operations may participate. Let $\Lambda =< \Lambda_{sorts}, \Lambda_{opns} >$ be a signature with $\Lambda_{sorts} \subseteq T_{sorts}$ and $\Lambda_{opns} \subseteq T_{opns}$. Then the *restriction* of the theory of T to Λ is

$$\mathsf{Th}(T)|_\Lambda := \{(t_0 = t_1) \in \mathsf{Th}(T) | t_0, t_1 \in \mathsf{terms}(\Lambda)\}.$$

The interpretation of an abstract type T is described using T_{sign}–algebras. A T_{sign}–algebra $\mathcal{A}$ is a collection of *carrier sets* $\mathcal{A}_C$ and a set of *operations* $\mathcal{A}_O$ on these carrier sets. For each sort in T_{sorts}, there is a carrier set in $\mathcal{A}_C$ and $\mathcal{A}_C$ contains no more sets. In a formula, this reads

$$\mathcal{A}_C = \{s^{\mathcal{A}} | s \in T_{sorts}\}.$$

Similarly, $\mathcal{A}_O$ consists of interpretations of the operation names. Here interpretations are also denoted by a superscript:

$$f^{\mathcal{A}} : s^{\mathcal{A}} \rightarrow t^{\mathcal{A}}$$

stands for the operation associated with the symbol $f : s \rightarrow t$ from T_{opns}.

In the usual way (e.g. [vD83]) one arrives at the interpretation of terms. For a ground term $t \in \mathsf{grterms}(T_{sign})_s$, $t^{\mathcal{A}} \in s^{\mathcal{A}}$ denotes its interpretation. To interprete non–ground terms, we need a function assigning a value to each variable in the language. Such a function is called a *valuation*. If v is a valuation, then $t^{\mathcal{A},v}$ ($t \in \mathsf{terms}(T_{sign})$) denotes the corresponding interpretation of t.

A T_{sign}–algebra is a *model* of the abstract type T iff it satisfies the laws of T, i.e. $\mathcal{A} \models T_{laws}$. $\mathcal{A}$ is an *initial model* of T iff the laws T_{laws} dictate all equalities in $\mathcal{A}$, and any value can be denoted by a ground term

$$\forall t_0, t_1 \in \mathsf{terms}(T_{sign}) : (t_0 = t_1) \in \mathsf{Th}(T) \Leftrightarrow \mathcal{A} \models (t_0 = t_1),$$

$$\forall s \in T_{sorts} : \forall a \in s^{\mathcal{A}} : \exists t \in \mathsf{grterms}(T_{sign})_s : t^{\mathcal{A}} = a.$$

In general, the existence of an initial model of T is not guaranteed [WPP*83]. In the present context, this problem can be ignored.

As in the case of the theory of a type T ($\mathsf{Th}(T)$), we will need a way of cutting portions from a T_{sign}–algebra $\mathcal{A}$. Let $\Lambda =< \Lambda_{sorts}, \Lambda_{opns} >$ be a signature with $\Lambda_{sorts} \subseteq T_{sorts}$ and $\Lambda_{opns} \subseteq T_{opns}$. If $\mathcal{A} =< \mathcal{A}_C, \mathcal{A}_O >$ is a T_{sign}–algebra then $\mathcal{A}|_\Lambda$ stands for the Λ–algebra $< \mathcal{A}'_C, \mathcal{A}'_O >$ with

$$\begin{aligned} \mathcal{A}'_C &:= \{s^{\mathcal{A}} | s \in \Lambda_{sorts}\} \\ \mathcal{A}'_O &:= \{f^{\mathcal{A}} | (f : s_0 \times s_1 \times \ldots \times s_{n-1} \rightarrow s_n) \in \Lambda_{opns}\} \end{aligned}$$

The algebra $\mathcal{A}|_\Lambda$ is called the Λ–*reduct* of $\mathcal{A}$.

2.3 Indeterminacy

The issue of indeterminate choice is a hotly debated topic in transformational programming. An indeterminate choice operator is a useful device in specification, but it may produce several undesirable effects unless it is used with extreme care [BMW85,Mor87]. For the purposes of this report, it is indispensable. The problem is that there are difficulties in treating the choice operator within the framework of abstract types. This topic is a research theme in its own right. Quoting from [Mol85]:

> "A deeper generalization, finally, would be the extension to nondeterminate operations in the algebras to cover also languages with nondeterminism. However, it is not clear how an adequate notion of homomorphism can be defined for such algebras and what axioms for nondeterminate algebras should look like."

Here, we will only list some characteristic properties of the indeterminate choice. Also, a relaxed way of talking about the semantics of expressions involving indeterminacy is discussed.

Let T be an abstract type. For any $s \in T_{sorts}$, $\Box_s$ denotes the indeterminate choice. Also, for each sort a constant $\bot_s$ is introduced. It stands for the choice from nothing, i.e. "undefined". The language of terms over T_{sign}, extended with these syntactical operators, will be denoted by $\mathsf{iterms}(T_{sign})$ ($\mathsf{igrterms}(T_{sign})$ for ground terms). The laws T_{laws} may use this language in the predicates.

The semantics of an indeterminate expression will be described by its set of possible values. The carrier sets in a model $\mathcal{A}$ of T, however, are still viewed as sets of objects, not as sets of sets of objects. The interpretation of terms and functions from the old language without indeterminacy is denoted by a superscript as before. As in [BBB*85] and [Mee86], the values are assumed to be present as constants in the old language $\mathsf{grterms}(T_{sign})$. The undefined value of sort s is also written using a superscript: $\bot_s^{\mathcal{A}}$.

The set of possible values of a term is called its *breadth*. It will be specified using the breadth function

$$\mathcal{B}_s^{\mathcal{A}} : \mathsf{igrterms}(T_{sign})_s \to \mathcal{P}(s^{\mathcal{A}}),\ (s \in T_{sorts}).$$

This function is characterized by the following equations.

(i) $\mathcal{B}_s^{\mathcal{A}}(\bot_s) = \emptyset$

(ii) $\forall t \in \mathsf{grterms}(T_{sign})_s : \mathcal{B}_s^{\mathcal{A}}(t) = \{t^{\mathcal{A}}\} - \{\bot_s\}$

(iii) $\forall t_0, t_1 \in \mathsf{igrterms}(T_{sign})_s : \mathcal{B}_s^{\mathcal{A}}(t_0 \Box_s t_1) = \mathcal{B}_s^{\mathcal{A}}(t_0) \cup \mathcal{B}_s^{\mathcal{A}}(t_1)$

(iv) For all $(f : u \rightarrow s)$ in T_{opns} :

$$\forall t \in \mathsf{igrterms}(T_{sign})_u : \mathcal{B}_s^{\mathcal{A}}(f\ t) = \bigcup\{\mathcal{B}_s^{\mathcal{A}}(f\ a) | a \in \mathcal{B}_u^{\mathcal{A}}(t)\}$$

Two terms $t_0, t_1 \in \mathsf{igrterms}(T_{sign})_s$ are said to be *equivalent* iff $\mathcal{B}_s^{\mathcal{A}}(t_0) = \mathcal{B}_s^{\mathcal{A}}(t_1)$. Indeed, this induces a congruence relation on $\mathsf{igrterms}(T_{sign})$ (see e.g. [MG83]), and considering terms as representatives of the congruence classes one may write $(t_0, t_1 \in \mathsf{igrterms}(T_{sign})_s)$

$$t_0 = t_1 \text{ iff } \mathcal{B}_s^{\mathcal{A}}(t_0) = \mathcal{B}_s^{\mathcal{A}}(t_1).$$

Slightly abusing terminology, we will say that the model $\mathcal{A}$ *satisfies* ($\models$) such equalities. According to the above properties of the breadth function, $\mathcal{A}$ satisfies the following features of indeterminate choice:

(i) $\forall t \in \mathsf{igrterms}(T_{sign})_s : t \Box_s \bot_s = \bot_s \Box_s t = t$

(ii) $\Box_s$ is associative, commutative and idempotent

(iii) For all $(f : u \rightarrow s)$ in T_{opns} :

$$\forall t_0, t_1 \in \mathsf{igrterms}(T_{sign})_u : (f\ t_0 \Box_u t_1) = (f\ t_0) \Box_s (f\ t_1).$$

The first property reflects the optimism of the approach: Choice between something and nothing will lead to something. It expresses the opposite of Murphy's law: If something might go right, it will. Indeed, we do not claim that the expressions are executable.

There is a price to these properties though, as will be illustrated in the following example.

Example 2
Let $(f : \mathsf{int} \rightarrow \mathsf{int})$ and $(\forall \mathsf{int}\ x : f\ x = x - x)$. What is the meaning of $f\ 0 \Box_{\mathsf{int}} 1$? The term $(0 \Box_{\mathsf{int}} 1)$ cannot be interpreted as an ordinary value from $\mathsf{int}^{\mathcal{A}}$, since that would give rise to undesirable equalities, as is

illustrated below. Using $\forall$–elimination (unfold) and property (iii) of $\Box_{\mathsf{int}}$, one may derive that:

$$f\ 0\Box_{\mathsf{int}}1 = (0\Box_{\mathsf{int}}1) - (0\Box_{\mathsf{int}}1) = -1\Box_{\mathsf{int}}0\Box_{\mathsf{int}}1.$$

On the other hand, one might derive that $f\ 0\Box_{\mathsf{int}}1 = 0$, and therefore

$$0 = -1\Box_{\mathsf{int}}0\Box_{\mathsf{int}}1.$$

This problem is often called "loss of referential transparency".

One solution is to prohibit $\forall$–elimination (unfolding) with indeterminate objects (see below) in the argument. This is quite restrictive, but for the present it will do.

A specific class of expressions is formed by those that denote a single value (possibly "undefined"). Such expressions are called determinate. A term $t \in \mathsf{igrterms}(T_{sign})_s$ is *determinate in* $\mathcal{A}$ iff $|\mathcal{B}_s^{\mathcal{A}}(t)| \leq 1$. A function $f : s \to u$ is *determinate in* $\mathcal{A}$ iff for any $t \in \mathsf{grterms}(T_{sign})_s$, $(f\ t)$ is determinate in $\mathcal{A}$. In reasoning about determinate functions it will be convenient if these operations are total. A function $f : s \to u$ is *total in* $\mathcal{A}$ iff for any $t \in \mathsf{grterms}(T_{sign})_s$, $\mathcal{B}_s^{\mathcal{A}}(f\ t) \neq \emptyset$. All functions are assumed to be total, unless explicitly stated otherwise. Note that the subject of partial functions was ignored in the exposition on abstract types. A possible approach may be found in [WPP*83].

2.4 Refinement

In the development of actual computer programs, indeterminacy has to be removed. Using the breadth function from the preceding section, one may characterize the *refinement relation* $\sqsupseteq_s^{\mathcal{A}}$ on terms of sort s. In turn, the refinement of functions can be pinned down.

(i) $\forall t_0, t_1 \in \mathsf{igrterms}(T_{sign})_s : t_0 \sqsupseteq_s^{\mathcal{A}} t_1$ iff $\mathcal{B}_s^{\mathcal{A}}(t_0) \supseteq \mathcal{B}_s^{\mathcal{A}}(t_1)$

(ii) For all $(f : s \to u)$, $(g : s \to u)$ in T_{opns} :

$$f \sqsupseteq_{s\to u}^{\mathcal{A}} g$$

iff $\forall t \in \mathsf{grterms}(T_{sign})_s : (f\ t) \sqsupseteq_u^{\mathcal{A}} (g\ t)$

Beware of the fact that any term of sort s refines to $\perp_s$; the relation $\sqsupseteq_s^{\mathcal{A}}$ does not coincide with "may be replaced by". Refinement is reflexive and transitive, hence refinement chains may be glued together. Another desirable feature is the local refinement of expressions. Let $t[e]$ denote a term of sort s with subexpression e (sort u). If for all terms e_0

$$e \sqsupseteq_u^{\mathcal{A}} e_0 \Rightarrow t[e] \sqsupseteq_s^{\mathcal{A}} t[e_0]$$

then $t[e]$ is *monotonic* with respect to the refinement relation. To validate the free application of local refinements, non–monotonic constructs are excluded from the expression language. This is a common restriction; it is also imposed in [BBB*85]. In the sequel, the subscripts and superscripts of $\mathcal{B}$, $\sqsupseteq$ and $\sqsubset$ are omitted. Likewise, we will say 'determinate' instead of 'determinate in $\mathcal{A}$' and 'total' instead of 'total in $\mathcal{A}$'.

Chapter 3

Theory

3.1 Manageable structures

The abstract types used for specification are hard to use in program development. Their general nature renders it difficult to formulate interesting transformation theorems. In this section, we seek to identify parts of abstract types that are apt to manipulation in algorithmics.

In [Mee86], morphic programming is applied to very simple data structures, which are built using only three operations: an embedding, a binary operator and a constant. Of course, the general concepts are applicable to more complex structure types [Mee87c]. For ease of exposition, our attention will be confined to the binary structures from the early days of morphic programming. A slight generalization is allowed, though: the embedding is not necessarily a singleton constructor and the constant is not always a unit element.

Definition 3 (*structure signature*)
Let T be an abstract type. A signature Σ is a *structure signature in* T if

(i) $\Sigma_{sorts} = \{d, s\} \subseteq T_{sorts}$

(ii) $\Sigma_{opns} = \{\hat{} : d \to s, \oplus : s \times s \to s, c :\to s\} \subseteq T_{opns}$.

The structure signature in definition 3 provides a view on the terms of sort s. Only terms that are constructed by the operations in the signature

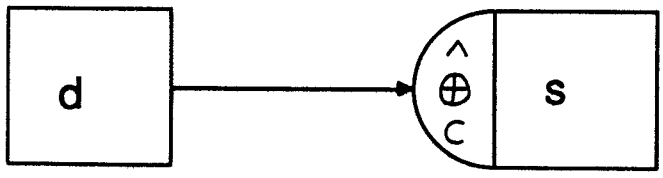

Figure 3.1: The structure signature provides a view on s.

are visible. Indeed, such terms are called visible; a precise definition will be given below. The vision metaphor is useful in the graphical illustration of results. A picture segment that is to represent a structure–signature–as–a–view is displayed in figure 3.1. Our notion of a 'view' is reminiscent of that in [Wad87].

Example 4
Consider the abstract type SEQU in example 1 (section 2.2).

$$\begin{aligned}\mathsf{listsign} :=< \{\mathsf{int}, \mathsf{iseq}\}, \{\ & [\_] : \mathsf{int} \to \mathsf{iseq}, \\ & \_ \# \_ : \mathsf{iseq} \times \mathsf{iseq} \to \mathsf{iseq}, \\ & [] :\to \mathsf{iseq}\} >\end{aligned}$$

is a structure signature in SEQU. There is a deviation from the definition above in the naming of the operation symbols. The conventional squiggles for list construction have been used. Likewise, conventional symbols for set construction are adopted throughout this paper.

Example 5
Yet another view on iseq is provided by

$$\begin{aligned}\mathsf{revlistsign} :=< \{\mathsf{int}, \mathsf{iseq}\}, \{\ & [\_] : \mathsf{int} \to \mathsf{iseq}, \\ & \_ \widetilde{\#} \_ : \mathsf{iseq} \times \mathsf{iseq} \to \mathsf{iseq}, \\ & [] :\to \mathsf{iseq}\} > .\end{aligned}$$

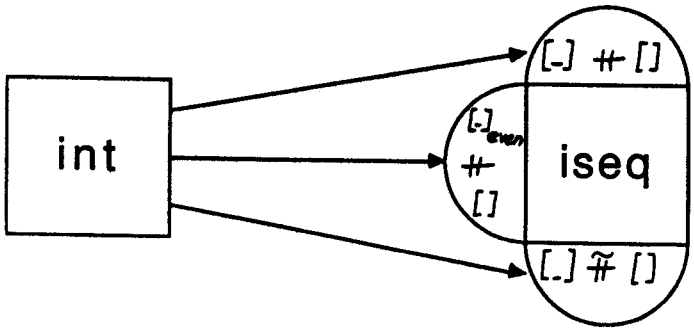

Figure 3.2: Different views on iseq.

This is also a structure signature in SEQU.

Example 6
Even the signature

$$\begin{aligned}\mathsf{evenlistsign} :=< \{\mathsf{int}, \mathsf{iseq}\}, \{\ & [\_]_{even} : \mathsf{int} \to \mathsf{iseq}, \\ & \_ +\!\!+ \_ : \mathsf{iseq} \times \mathsf{iseq} \to \mathsf{iseq}, \\ & [] :\to \mathsf{iseq}\} >\end{aligned}$$

yields a view on iseq. The different views from examples 4, 5 and 6 are depicted in figure 3.2.

In general, the existence of an initial model of an abstract type is not guaranteed. Much effort has been succesfully invested in the formulation of sufficient conditions implying this desirable property [WPP*83]. Here, it will be tacitly assumed that all abstract types introduced actually have an initial model.

Definition 7 (*structure*)
Let T be an abstract type, Σ a structure signature in T. Let $\mathcal{A}$ be an initial model of T. The Σ–reduct $\mathcal{A}|_\Sigma$ is the *structure of type* Σ *in* $\mathcal{A}$.

Example 8
The structure signature listsign was introduced in example 4. *Lists* is the structure of type listsign in the initial model of SEQU. It represents the familiar lists.

Example 9
The structure signature evenlistsign was introduced in example 6. *Evenlists* is the structure of type evenlistsign in the initial model of SEQU. It represents the lists with even numbers as elements.

The carrier $d^{\mathcal{A}}$ is called the *domain*, $s^{\mathcal{A}}$ the *main carrier*, $\hat{}^{\mathcal{A}}$ the *embedding*, $\oplus^{\mathcal{A}}$ the *operator* and $c^{\mathcal{A}}$ the *constant* of the structure. A structure as defined here is nothing but a cutting from the initial model. Such structures are not always as easy to handle as one might wish. It could be that the properties of the operations depend "too much" on other parts of the abstract type. To investigate this, some further definitions are needed.

Convention 10 (*abstract type and its initial model*)
In the sequel, T denotes an abstract type, and $\mathcal{A}$ is its initial model.

Visible terms are those which denote values in $s^{\mathcal{A}}$ that could be constructed using the operations in the structure. Their general form is depicted in figure 3.3.

Definition 11 (*visible terms*)
Let $\Sigma :=< \{d, s\}, \{\hat{} : d \to s, \oplus : s \times s \to s, c :\to s\} >$ be a structure signature in T. The *set of* Σ*-visible terms,* $\mathsf{vterms}(\mathsf{T}, \Sigma)$, is the smallest set of ground terms S satisfying:

(i) (a) $c \in S$
(b) $\forall a \in \mathsf{grterms}(\mathsf{T}_{sign})_d : (\hat{}\, a) \in S$
(ii) $\forall t_0, t_1 \in S : t_0 \oplus t_1 \in S$.

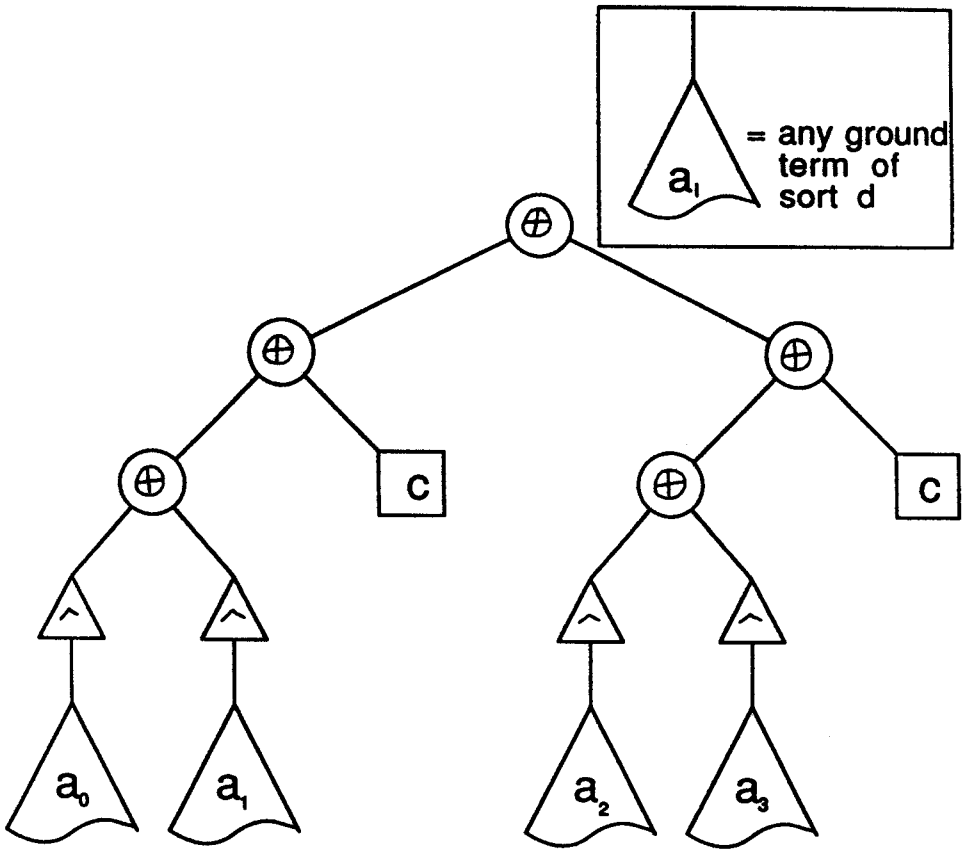

Figure 3.3: Visible terms.

Definition 12 (*visible subset of main carrier*)
Let $\Sigma :=< \{d, s\}, \{\hat{} : d \to s, \oplus : s \times s \to s, c :\to s\} >$ be a structure signature in T. The *Σ–visible subset of $s^{\mathcal{A}}$*, $s^{\mathcal{A}}|_{\Sigma}$, is

$$s^{\mathcal{A}}|_{\Sigma} := \{t^{\mathcal{A}} | t \in \mathsf{vterms}(\mathsf{T}, \Sigma)\}.$$

Convention 13 (*omission of signature in "visible"*)
If the structure signature Σ is clear from the context, it may be omitted in "Σ–visible".

The elements from the visible subset are said to be *visible values*. Likewise, values in $(s^{\mathcal{A}} - s^{\mathcal{A}}|_{\Sigma})$ (that cannot be denoted by a visible term) are called *invisible*. Collectively, invisible values are known as *junk* (see e.g. [MG83]). Note that both the visible subset and the junk may vary according to the view provided by a specific choice of structure signature.

Example 14
In *Evenlists* (example 9), singletons with an uneven element are invisible. It is important to realize that they are present in the main carrier ($\mathsf{iseq}^{\mathcal{A}}$) though.

Proposition 15 (*visible subset in terms of structure signature*)
Let $\Sigma :=< \{d, s\}, \{\hat{} : d \to s, \oplus : s \times s \to s, c :\to s\} >$ be a structure signature in T. Let X_d be the set of variables (of sort d) occurring in terms over T_{sign}. Then

$$s^{\mathcal{A}}|_{\Sigma} = \{t^{(\mathcal{A},v)} |\ t \in \mathsf{terms}(\Sigma)_s, \text{all variables in } t \text{ from } X_d, \\ v \text{ a valuation } X_d \to d^{\mathcal{A}}\}.$$

Proof: Immediate from the fact that the initial model $\mathcal{A}$, and hence $d^{\mathcal{A}}$, contains no junk as a whole. □

The use of induction to prove facts on visible terms (and values) is so familiar that one hardly thinks about its validity. The technique is safe here, and the principle is stated as a fact. A proof may be found in any introduction to abstract types (e.g. [MG83]).

Fact 16
Let $\Sigma :=< \{d, s\}, \{\hat{} : d \to s, \oplus : s \times s \to s, c :\to s\} >$ be a structure signature in T. Let ϕ be a unary predicate over $\mathsf{terms}(\mathsf{T}_{sign})$. If $\mathcal{A}$ satisfies

(i) $\phi\, c$

(ii) $\forall a \in \mathsf{grterms}(\mathsf{T}_{sign})_d : \phi\, \hat{a}$

(iii) $\forall t_0, t_1 \in \mathsf{vterms}(\mathsf{T}, \Sigma) : (\phi t_0) \wedge (\phi t_1) \rightarrow \phi(t_0 \oplus t_1)$

then $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma) : \mathcal{A} \models (\phi\, t)$.

The present aim is to regard a structure as an algebra over the domain $d^{\mathcal{A}}$. The concept of an 'algebra over a domain', omnipresent in computer science and mathematics, was formulated in [Zan87]. A precise definition will be given below. The basic idea is that it should be possible to talk about sets, bags and sequences just as we do in classical mathematics about groups and rings, irrespective of the idiosyncrasies of the domain. In colloquial terms, it is the outside, the surface that matters. The following definitions pin these ideas down formally.

Definition 17 (*surface signature*)
Let $\Sigma :=< \{d, s\}, \{\hat{} : d \to s, \oplus : s \times s \to s, c :\to s\} >$ be a structure signature in T. The *surface signature of* Σ is

$$< \{s\}, \{\oplus : s \times s \to s, c :\to s\} > .$$

Convention 18 (*shorthand for declaration of structures*)

(i) "$\Sigma : (d, s, \hat{}, \oplus, c)$" stands for
"Let $\Sigma :=< \{d, s\}, \{\hat{} : d \to s, \oplus : s \times s \to s, c :\to s\} >$ be a structure signature in T."

(ii) "$\mathcal{S} : \Sigma$" stands for "Let $\mathcal{S}$ be the structure of type Σ in $\mathcal{A}$, the initial model of T."

In the manipulation of terms of sort s, one is primarily interested in laws expressed as equations over the surface signature, since these properties will be generally applicable. For example, in calculating with lists, the following laws are most important:

$$\begin{array}{lcl} x +\!\!+ [\,] & = & x \\ [\,] +\!\!+ x & = & x \\ (x +\!\!+ y) +\!\!+ z & = & x +\!\!+ (y +\!\!+ z). \end{array}$$

The algebraic richness of a structure contains all such properties and nothing more.

Definition 19 (*algebraic richness*)
Let $\Sigma : (d, s, \hat{}, \oplus, c)$, $\mathcal{S} : \Sigma$. Let Λ be the surface signature of Σ. The *algebraic richness of* $\mathcal{S}$, $\$(\mathcal{S})$, is the set of equations:

$$\$(\mathcal{S}) := \{t_0{=}t_1 | t_0, t_1 \in \mathsf{terms}(\Lambda),\ \mathcal{S} \models t_0{=}t_1\}.$$

Proposition 20 (*algebraic richness as restriction of theory of* T)
Let $\Sigma : (d, s, \hat{}, \oplus, c)$, $\mathcal{S} : \Sigma$. Let Λ be the surface signature of Σ.

$$\$(\mathcal{S}) = \mathsf{Th}(\mathsf{T})|_\Lambda.$$

Proof: $\mathcal{A}$ is an initial model of T, hence $\forall t_0, t_1 \in \mathsf{terms}(\mathsf{T}_{sign}) : \mathcal{A} \models t_0{=}t_1 \Leftrightarrow \mathsf{T} \vdash t_0{=}t_1$. In particular, this property holds for terms in $\mathsf{terms}(\Lambda)$. For such terms, $\mathcal{A} \models t_0{=}t_1 \Leftrightarrow \mathcal{S} \models t_0{=}t_1$. □

The question is: Does the algebraic richness tell us all we need to know about terms of sort s, or are there 'unexpected' equalities, depending on the semantics of the embedding? It is difficult to formulate a definition capturing the structures that do not exhibit this undesirable phenomenon. A sufficient condition is that the structure is an initial algebra over the domain $d^{\mathcal{A}}$, with the algebraic richness as laws.

Definition 21 (*L–initial algebra over the domain*)
Let $\Sigma : (d, s, \hat{}, \oplus, c)$, $\mathcal{S} : \Sigma$. Let L be a set of equations over the surface–signature of Σ. $\mathcal{S}$ is an *L–initial algebra over* $d^{\mathcal{A}}$ if

(i) $s^{\mathcal{A}}|_{\Sigma} = s^{\mathcal{A}}$

(ii) $\forall t_0, t_1 \in \mathsf{terms}(\Sigma)_s : \mathcal{S} \models t_0 = t_1 \Leftrightarrow L \vdash t_0 = t_1$.

Definition 22 (*manageable structure*)
Let $\Sigma : (d, s, \hat{}, \oplus, c)$, $\mathcal{S} : \Sigma$. $\mathcal{S}$ is *manageable* if $\mathcal{S}$ is an $\$(\mathcal{S})$–initial algebra over $d^{\mathcal{A}}$.

Example 23
Consider the structure *Lists* (with integer elements) from example 8. Its signature is

$$\begin{aligned}\mathsf{listsign} :=< \{\mathsf{int}, \mathsf{iseq}\}, \{\ & [\_] : \mathsf{int} \to \mathsf{iseq}, \\ & \_ \# \_ : \mathsf{iseq} \times \mathsf{iseq} \to \mathsf{iseq}, \\ & [\,] :\to \mathsf{iseq}\} > .\end{aligned}$$

The surface signature is

$$\begin{aligned}\Lambda :=< \{\mathsf{iseq}\}, \{\ & \_ \# \_ : \mathsf{iseq} \times \mathsf{iseq} \to \mathsf{iseq}, \\ & [\,] :\to \mathsf{iseq}\} > .\end{aligned}$$

From the laws in the enclosing abstract type SEQU, one may conclude that the algebraic richness consists of

$$\begin{aligned}\{\ & x \# [\,] && = x, \\ & [\,] \# x && = x, \\ & (x \# y) \# z && = x \# (y \# z)\},\end{aligned}$$

and all equalities in terms(Λ) that may be deduced from these equations. Indeed, two lists of integers are equal if and only if this may be proved from the three laws above (and equalities between the elements). Also, any list may be constructed by the operations in the structure. As a consequence, *Lists*, the lists with integer elements, is a manageable structure. Similarly, familiar structures like sets of integers, binary trees with character leaves and bags with purchases are manageable.

Example 24
An instance of a structure that is not manageable is provided by *Evenlists* from example 9. For easy reference, the structure signature is repeated here.

$$\begin{aligned}\textsf{evenlistsign} :=< \{\textsf{int}, \textsf{iseq}\}, \{\ & [\_]even : \textsf{int} \to \textsf{iseq}, \\ & \_ +\!\!+ \_ : \textsf{iseq} \times \textsf{iseq} \to \textsf{iseq}, \\ & [\,] :\to \textsf{iseq}\} > .\end{aligned}$$

The surface signature and algebraic richness are the same as for the structure *Lists* above. Recall the law defining $[\_]even$ in SEQU (example 1)

$$\begin{aligned}\forall \textsf{int}\ n : [n]even = \quad & even\, n \quad \to \ [n] \\ \square \quad & \neg even\, n \quad \to \ [\,].\end{aligned}$$

We have e.g. $[3]even = [\,]$. As was already noted in example 14, the value $[3]^{\mathcal{A}}$ is not visible. Hence *Evenlists* is not manageable.

3.2 Homomorphisms

The importance of maps that preserve the structure of terms has been convincingly demonstrated in all papers on algorithmics. Many non–trivial transformations may be succinctly formulated as identities involving homomorphisms. There are certain conditions to be satisfied to ensure that such a map exists between two given structures. These constraints are investigated in the present section.

For our purposes, only a restricted kind of homomorphism is needed: those between structures in the initial model $\mathcal{A}$ of the abstract type T. Whenever the term "homomorphism" is used in this report, we refer to the restricted form defined below.

Definition 25 (*homomorphism*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (d, u, \tilde{}, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$.
A mapping $\phi : s^{\mathcal{A}} \rightarrow u^{\mathcal{A}}$ is a *homomorphism from* $\mathcal{S}_0$ *to* $\mathcal{S}_1$ if

(i) $\phi\ c_0{}^{\mathcal{A}} = c_1{}^{\mathcal{A}}$

(ii) $\phi \circ \hat{}^{\mathcal{A}} = \tilde{}^{\mathcal{A}}$

(iii) $\forall x, y \in s^{\mathcal{A}} : \phi(x \oplus^{\mathcal{A}} y) = (\phi\ x) \otimes^{\mathcal{A}} (\phi\ y)$.

One may translate any term from the language of one structure into that of another. The point is: will terms, which evaluate to the same value, when mapped to the other structure still evaluate to the same value? A good start to solve the problem is the definition of term translations.

Definition 26 (*term translation*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\Sigma_1 : (d, u, \tilde{}, \otimes, c_1)$.
Let $X_s = \{x_0, x_1, \ldots\}$ be the set of variables of sort s. Let $X_u = \{y_0, y_1, \ldots\}$ be the set of variables of sort u. Let $\mathsf{vtermsx}(\mathsf{T}, \Sigma_0)$ ($\mathsf{vtermsx}(\mathsf{T}, \Sigma_1)$) be the set of visible terms with variables from X_s (X_u).
The *term translation from* Σ_0 *to* Σ_1,
$\mathsf{TR}_{\Sigma_0 \rightarrow \Sigma_1} : \mathsf{vtermsx}(\mathsf{T}, \Sigma_0) \rightarrow \mathsf{vtermsx}(\mathsf{T}, \Sigma_1)$, is defined by:

(i) $\forall i \in \{0, 1, 2, \ldots\} : \mathsf{TR}_{\Sigma_0 \rightarrow \Sigma_1}\ x_i := y_i$

(ii) $\mathsf{TR}_{\Sigma_0 \rightarrow \Sigma_1}\ c_0 := c_1$

(iii) $\forall a \in \mathsf{grterms}(\mathsf{T}_{sign})_d : \mathsf{TR}_{\Sigma_0 \rightarrow \Sigma_1}\ \hat{}a := \tilde{}a$

(iv) $\forall t_0, t_1 \in \mathsf{vtermsx}(\mathsf{T}, \Sigma_0)$:

$$\mathsf{TR}_{\Sigma_0 \rightarrow \Sigma_1}(t_0 \oplus t_1) := (\mathsf{TR}_{\Sigma_0 \rightarrow \Sigma_1}\ t_0) \otimes (\mathsf{TR}_{\Sigma_0 \rightarrow \Sigma_1}\ t_1).$$

The vision metaphor (a signature is a view on a sort) will be useful in talking about term translations, too. A picture segment denoting the translation from a view on one sort to a view on another is displayed in figure 3.4.

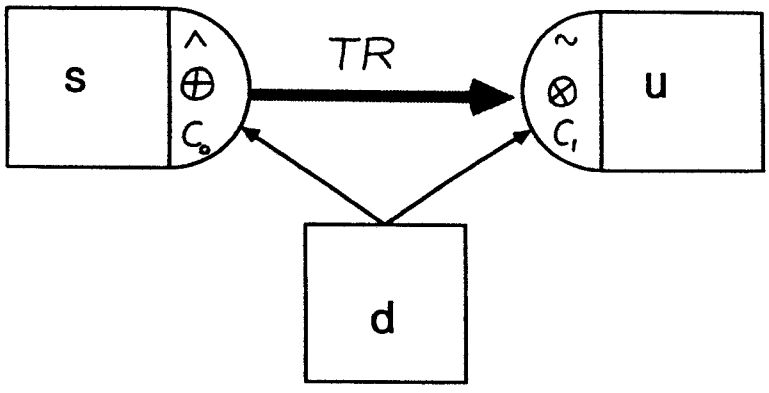

Figure 3.4: Graphical representation of a term translation.

Convention 27 (*omission of types in term translation*)
If in formulas involving $\mathsf{TR}_{\Sigma_0 \to \Sigma_1}$ both structure signatures Σ_0 and Σ_1 are clear from the context, then they may be omitted: $\mathsf{TR} := \mathsf{TR}_{\Sigma_0 \to \Sigma_1}$.

In translating terms from the language of one structure to that of another, the equality relation should be preserved. It stands to reason to compare the structures with respect to their algebraic richness. Intuitively, the source structure should be the poorer one in identities.

Definition 28 (*poorer*)
Let Σ_0 and Σ_1 be structure signatures in T. Let $\mathcal{S}_0 : \Sigma_0$ and $\mathcal{S}_1 : \Sigma_1$. $\mathcal{S}_0$ *is poorer than* $\mathcal{S}_1$, $(\mathcal{S}_0 \preceq \mathcal{S}_1)$, if

$$\{(\mathsf{TR}\, t_0)=(\mathsf{TR}\, t_1) | t_0=t_1 \in \$(\mathcal{S}_0)\} \subseteq \$(\mathcal{S}_1).$$

As is clear from the definition, "not richer" would be a more accurate term. However, the theory is already difficult to pronounce as it is, which seems sufficient reason to allow for this slight anomaly in nomenclature.

Proposition 29 ($\preceq$ *is reflexive and transitive*)
Let $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2$ be arbitrary structures in $\mathcal{A}$.

(i) $\mathcal{S}_0 \preceq \mathcal{S}_0$

(ii) $\mathcal{S}_0 \preceq \mathcal{S}_1$ and $\mathcal{S}_1 \preceq \mathcal{S}_2$ then $\mathcal{S}_0 \preceq \mathcal{S}_2$.

Proof: Note that term translations satisfy the following properties

$$\begin{aligned} \mathsf{TR}_{\Sigma\to\Sigma} &= \mathbf{id} \\ \mathsf{TR}_{\Sigma_1\to\Sigma_2} \circ \mathsf{TR}_{\Sigma_0\to\Sigma_1} &= \mathsf{TR}_{\Sigma_0\to\Sigma_2}. \end{aligned}$$

The proposition follows from these facts and reflexivity and transitivity of $\subseteq$. $\square$

For manageable structures, the intuition expressed above is right. There is only a well–defined homomorphism from $\mathcal{S}_0$ to $\mathcal{S}_1$ if $\mathcal{S}_0$ satisfies no more identities then $\mathcal{S}_1$. To prove this, we will need the following simple fact on the initiality of algebras and the existence of homomorphisms (see e.g. [MG83,Zan87]).

Fact 30
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (d, u, \tilde{}, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$.
Let L be a set of equations over the surface–signature of Σ_0. If

(i) $\mathcal{S}_0$ is an L–initial algebra over $d^{\mathcal{A}}$

(ii) $\mathcal{S}_1 \models \{(\mathsf{TR}\ t_0) = (\mathsf{TR}\ t_1) | (t_0 = t_1) \in L\}$,

then there is exactly one homomorphism from $\mathcal{S}_0$ to $\mathcal{S}_1$.

Theorem 31 (*projection into a richer structure*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (d, u, \tilde{}, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$.
If $\mathcal{S}_0$ is manageable then the following statements are equivalent:

(i) $\mathcal{S}_0 \preceq \mathcal{S}_1$

(ii) There is exactly one homomorphism from $\mathcal{S}_0$ to $\mathcal{S}_1$.

Proof: Let $\mathcal{S}_0$ be manageable.

($\Rightarrow$) Let $\mathcal{S}_0$ be poorer than $\mathcal{S}_1$, $\mathcal{S}_0 \preceq \mathcal{S}_1$. By definition 28, we have

$$\{(\mathsf{TR}\ t_0) = (\mathsf{TR}\ t_1) | (t_0 = t_1) \in \$(\mathcal{S}_0)\} \subseteq \$(\mathcal{S}_1).$$

According to the definition of algebraic richness (definition 19), this implies

$$\mathcal{S}_1 \models \{(\mathsf{TR}\ t_0) = (\mathsf{TR}\ t_1) | (t_0 = t_1) \in \$(\mathcal{S}_0)\}.$$

By the given that $\mathcal{S}_0$ is manageable (definition 22), $\mathcal{S}_0$ is a $\$(\mathcal{S}_0)$–initial algebra over $d^{\mathcal{A}}$. Therefore, by fact 30 there is exactly one homomorphism from $\mathcal{S}_0$ to $\mathcal{S}_1$.

($\Leftarrow$) Let ϕ be the unique homomorphism from $\mathcal{S}_0$ to $\mathcal{S}_1$. By induction (fact 16) it is easily shown that

$$\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : (\mathsf{TR}\ t)^{\mathcal{A}} = \phi\, t^{\mathcal{A}}.$$

Assume that there is some equation $(t_0 = t_1) \in \$(\mathcal{S}_0)$, with

$$\mathcal{S}_1 \not\models (\mathsf{TR}\ t_0) = (\mathsf{TR}\ t_1).$$

Then there are ground terms $t'_0, t'_1 \in \mathsf{vterms}(\mathsf{T}, \Sigma)$ such that

$$t'^{\mathcal{A}}_0 = t'^{\mathcal{A}}_1,$$

but $(\mathsf{TR}\ t'_0)^{\mathcal{A}} \neq (\mathsf{TR}\ t'_1)^{\mathcal{A}}$, so (see above) $\phi\, t'^{\mathcal{A}}_0 \neq \phi\, t'^{\mathcal{A}}_1$. This is in contradiction with the fact that ϕ is well–defined. One may conclude that

$$\{(\mathsf{TR}\ t_0) = (\mathsf{TR}\ t_1) | (t_0 = t_1) \in \$(\mathcal{S}_0)\} \subseteq \$(\mathcal{S}_1),$$

i.e. $\mathcal{S}_0 \preceq \mathcal{S}_1$.

□

Often, a syntactic denotation for a 'real' function on the carrier sets of $\mathcal{A}$ is absent in the abstract type T. For example, in the projection theorem above, there may be no function symbol corresponding to the homomorphism. We will tacitly assume that such symbols (along with the defining equations) may be added to the signature whenever convenient. Formally, one ought to check that such extensions of the type do not cause "confusion" or introduce "junk" in the initial model $\mathcal{A}$ [MG83]. We will not introduce new sorts in the abstract type.

Convention 32 (*extension of the abstract type*)
The abstract type T is extended with new operations whenever convenient. It will be tacitly assumed that this does not cause confusion or introduce junk in the initial model $\mathcal{A}$ of T.

In functional programming, two kinds of homomorphism have become extremely popular. First, the inserted–in, often named a reduce. An operator is inserted between all elements of its argument. An example is the sum of a list of integers:

$$+/[1,3,5,2] = 1 + 3 + 5 + 2 = 11.$$

The second celebrity is the apply–to–all. It is commonly known as the 'map'. Here a function is applied to all elements of its argument:

$$(+1) * [1,3,5,2] = [1+1, 3+1, 5+1, 2+1] = [2,4,6,3].$$

Proposition 33 (*inserted–in*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (e, d, \tilde{}, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$.
If

(i) $\mathcal{S}_0$ is manageable

(ii) $\mathcal{S}_0 \preceq \mathcal{S}_1$

then there is exactly one function, the *inserted–in from* $\mathcal{S}_0$ *to* $\mathcal{S}_1$, $\otimes/^{\mathcal{A}} : s^{\mathcal{A}} \to d^{\mathcal{A}}$ such that $\mathcal{A}$ satisfies:

(a) $\otimes/\ c_0 = c_1$

(b) $\forall d\ a : \otimes/\ \hat{a} = a$

(c) $\forall s\ x, y : \otimes/(x \oplus y) = (\otimes/x) \otimes (\otimes/y)$.

Proof: Let (i) and (ii) hold. Let $\Sigma_2 : (d, d, \mathbf{id}, \otimes, c_1)$, $\mathcal{S}_2 : \Sigma_2$. The situation is depicted in figure 3.5. Obviously: $\$(\mathcal{S}_1) = \$(\mathcal{S}_2)$, hence by (ii) and proposition 29, $\mathcal{S}_0 \preceq \mathcal{S}_2$. Using (i), theorem 31 yields the desired result. □

There is a possible pitfall in this proposition. The inserted–in could yield values that are not in the visible subset $d^{\mathcal{A}}|_{\Sigma_1}$.

Example 34
Let $[\alpha]$ denote the sort of lists with elements of sort α. For instance [[int]] stands for the lists of lists of integers and [int] corresponds to iseq in example 1. Let

$$\Sigma_0 : ([\mathsf{int}], [[\mathsf{int}]], [\_]_{[\mathsf{int}]}, +\!\!+_{[[\mathsf{int}]]}, [\,]_{[[\mathsf{int}]]}),\ \mathcal{S}_0 : \Sigma_0,$$

$$\Sigma_1 := \mathsf{evenlistsign},\ \mathcal{S}_1 := \mathcal{E}\mathit{venlists}.$$

The latter structure of lists of even integers was introduced in example 9. According to the proposition,

$$+\!\!+/ : [[\mathsf{int}]] \to \mathsf{iseq}$$

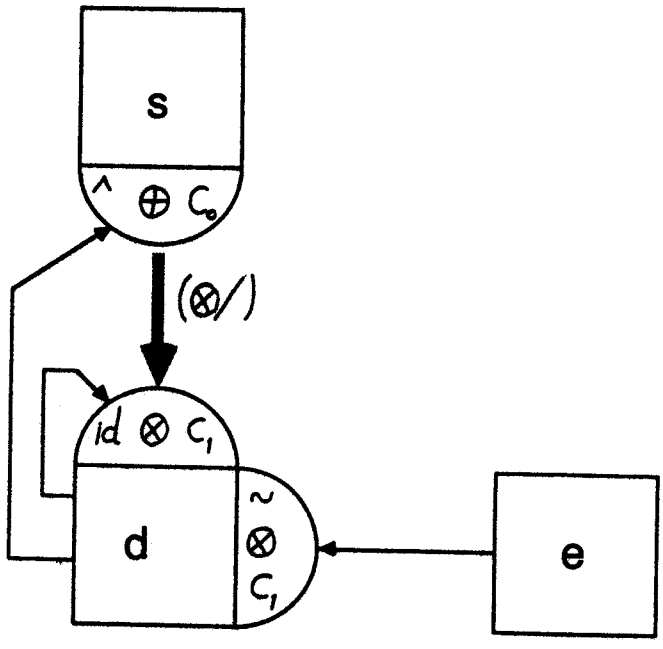

Figure 3.5: Inserted–in.

is a well–defined function. It satisfies

$$+\!\!+/[[3]] = [3].$$

However, $[3]^{\mathcal{A}}$ is not a visible value in *Evenlists* (example 14). This explains why it says in the proposition

$$\otimes/^{\mathcal{A}} : s^{\mathcal{A}} \to d^{\mathcal{A}},$$

instead of the more natural

$$\otimes/^{\mathcal{A}} : s^{\mathcal{A}} \to d^{\mathcal{A}}|_{\Sigma_1}.$$

Proposition 35 (*apply–to–all*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (e, u, \tilde{}, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$.
Let $f : d \to e$. If

(i) $\mathcal{S}_0$ is manageable

(ii) $\mathcal{S}_0 \preceq \mathcal{S}_1$

then there is exactly one function, the *apply–to–all from* $\mathcal{S}_0$ *to* $\mathcal{S}_1$, $(f*)^{\mathcal{A}} : s^{\mathcal{A}} \to t^{\mathcal{A}}$ such that $\mathcal{A}$ satisfies:

(a) $(f*)c_0 = c_1$

(b) $\forall d\ a : (f*)\hat{a} = \tilde{}fa$

(c) $\forall s\ x, y : (f*)(x \oplus y) = ((f*)x) \otimes ((f*)y)$.

Proof: Let (i) and (ii) hold. Let $\Sigma_2 : (d, u, \tilde{}\circ f, \otimes, c_1)$, $\mathcal{S}_2 : \Sigma_2$. The situation is depicted in figure 3.6. Obviously: $\$(\mathcal{S}_1) = \$(\mathcal{S}_2)$, hence by (ii) and proposition 29, $\mathcal{S}_0 \preceq \mathcal{S}_2$. Using (i), theorem 31 yields the desired result. □

An apply–to–all is meant to be used between two similar structures, e.g. lists of integers and lists of characters. It should not involve a change of algebraic richness. The apply–to–all introduced here is more general than the usual 'map'–function, as is exemplified below.

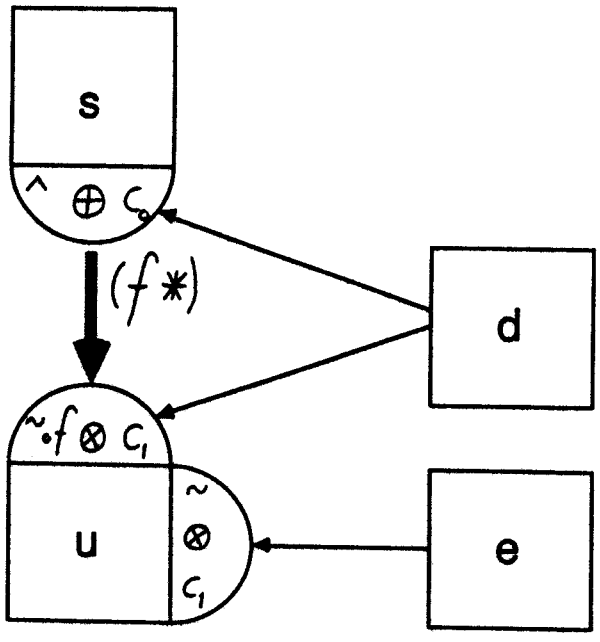

Figure 3.6: Apply–to–all.

Example 36
Let $\mathcal{L}ists$ be the conventional structure of lists of integers as in example 4. Let

$$\Sigma_1 \ : \ (\mathsf{int}, \mathsf{intset}, \{\_\}, \cup, \emptyset), \ \mathcal{S}ets : \Sigma_1$$

be the conventional structure of sets of integers. $(\mathbf{id}*)$ is an apply–to–all from $\mathcal{L}ists$ to $\mathcal{S}ets$. It turns a sequence of numbers into a set. Usually, this is written as $\cup/\{\_\}*$, employing the intermediate structure of lists of sets of numbers.

Because of the resulting ambiguities in expressions involving '$*$', the following convention is adopted.

Convention 37 (*restricted use of apply–to–all*)
Proposition 35 is only used if $\$(\mathcal{S}_0) = \$(\mathcal{S}_1)$ modulo renaming.

By now, the seasoned algorithmician may long to see the homomorphism lemma pop up. This lemma states that any homomorphism from a manageable structure to another may be decomposed into an inserted–in and an apply–to–all. However, in the present context, the existence of the intermediate structure needed to express this fact cannot be guaranteed. It could be missing in the initial model $\mathcal{A}$. An illustration of the result may be found in figure 3.7.

Under the assumption that all structures involved are present, one could prove the promotion laws at this point, using theorem 1. As mentioned in the introduction, they are simple corollaries to a theorem on the composition of divide–and–rule algorithms in section 3.4, so their discussion is will be postponed until that point.

3.3 Indeterminate insertion

In this section, we will discuss what happens if the constraints on algebraic richness are dropped in functions defined by term translation. No doubt, the result is an indeterminate map. Still, it is a special kind of indeterminacy that might exhibit interesting features. Here, the phenomenon is discussed briefly — a preparatory jump on the diving–board before we immerse ourselves in the general divide–and–rule algorithms. To give an

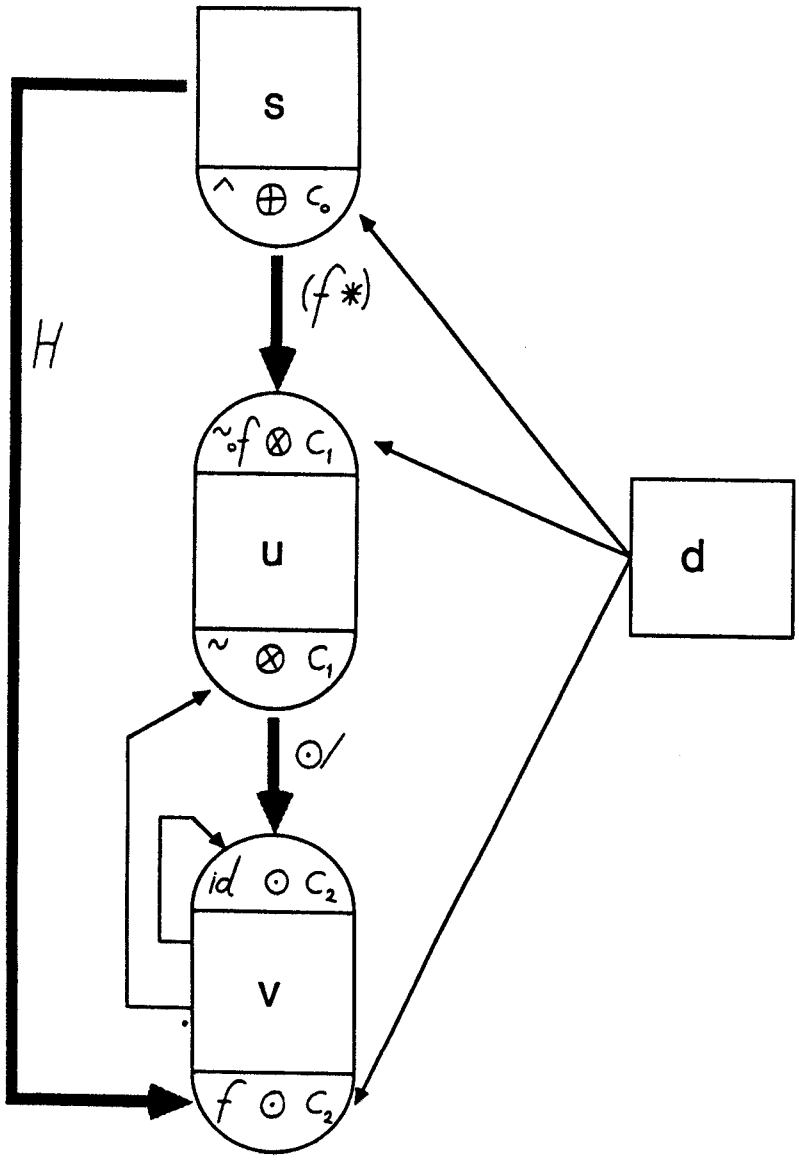

Figure 3.7: The homomorphism lemma: $H = \odot/f*$.

idea of what lies before us, consider the indeterminate map *isum*, with the following characteristics:

$$\begin{array}{lcl} \mathit{isum}\,\emptyset & = & 0 \\ \mathit{isum}\,\{a\} & \sqsupseteq & a \\ \mathit{isum}\,x \cup y & = & \mathbf{some}\{(\mathit{isum}\,s) + (\mathit{isum}\,t) | s \cup t = x \cup y\}. \end{array}$$

The indeterminacy lies in the fact that $\cup$ satisfies associativity, commutativity and idempotency, whereas $+$ is only associative and commutative. Nevertheless, $(\mathit{isum}\ \{2\})$ is a sensible expression. It specifies any non–zero multiple of 2. This kind of specification could be a useful device. Also, such indeterminate functions may be used in reasoning about generalized inverses. This point will be amplified in the next section (proposition 65). An 'indeterminate inserted–in' will be defined by its breadth. As is clear from the characterization of *isum*, one needs some way of talking about "all equal terms".

Definition 38 (*congruence class*)
Let Σ be a structure signature in T. Let $t \in \mathsf{vterms}(\mathsf{T}, \Sigma)$. The *congruence class of* t, $[t]^{\Sigma}$, is

$$[t]^{\Sigma} := \{t' | t' \in \mathsf{vterms}(\mathsf{T}, \Sigma), t^{\mathcal{A}} = t'^{\mathcal{A}}\}.$$

Convention 39 (*omission of structure signature in congruence class*)
If the structure signature Σ is clear from the context (usually quantification over a set of terms) it may be omitted in $[t]^{\Sigma}$:

$$[t] := [t]^{\Sigma}.$$

The next proposition is a restatement of the projection theorem (theorem 31). It clarifies the name of that result: finer congruence classes are projected into coarser ones.

Proposition 40 (*richness and granularity of congruence classes*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (d, u, \tilde{}, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$.
If $\mathcal{S}_0$ is manageable then the following statements are equivalent:

(i) $\mathcal{S}_0 \preceq \mathcal{S}_1$

(ii) $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : \{\mathsf{TR}\ t' | t' \in [t]\} \subseteq [\mathsf{TR}\ t]$.

Proof: Let $\mathcal{S}_0$ be manageable.

($\Rightarrow$) $\mathcal{S}_0 \preceq \mathcal{S}_1$. Then there is exactly one homomorphism $h^{\mathcal{A}}$ from $\mathcal{S}_0$ to $\mathcal{S}_1$ (theorem 31), hence by $(h\ t)^{\mathcal{A}} = (\mathsf{TR}\ t)^{\mathcal{A}}$

$$\forall t_0, t_1 \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : t_0{}^{\mathcal{A}} = t_1{}^{\mathcal{A}} \Rightarrow (\mathsf{TR}\ t_0)^{\mathcal{A}} = (\mathsf{TR}\ t_1)^{\mathcal{A}}.$$

From the definition of congruence classes (definition 38), one may conclude that

$$\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : \{\mathsf{TR}\ t' | t' \in [t]\} \subseteq [\mathsf{TR}\ t].$$

($\Leftarrow$) $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : \{\mathsf{TR}\ t' | t' \in [t]\} \subseteq [\mathsf{TR}\ t]$. Let $h^{\mathcal{A}} : s^{\mathcal{A}} \rightarrow u^{\mathcal{A}}|_{\Sigma_1}$ be a function such that $\mathcal{A}$ satisfies:

(a) $h\ c_0 = c_1$

(b) $\forall d\ a : h\ \widehat{a} = \widetilde{a}$

(c) $\forall s\ x, y : h(x \oplus y) = (h\ x) \otimes (h\ y)$.

By the assumption and the fact that

$$\forall t \in \mathsf{vterms}(T, \Sigma_0) : (h\ t)^{\mathcal{A}} = (\mathsf{TR}\ t)^{\mathcal{A}},$$

$h^{\mathcal{A}}$ is a unique homomorphism from $\mathcal{S}_0$ to $\mathcal{S}_1$. The projection theorem (theorem 31) yields $\mathcal{S}_0 \preceq \mathcal{S}_1$.

□

The neat definition of indeterminate inserted–in is an easy task by now. It boils down to specifying the translation of all terms representing the actual argument. Indeterminate inserted–in gives rise to expressions with an infinite breadth. The semantic problems connected with this non–determinism are left alone for the moment. It is conjectured without proof that indeterminate inserted–in is monotonic with respect to expression refinement. A picture of the sorts involved in the definition is provided in figure 3.8

Definition 41 (*indeterminate inserted–in*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0, \Sigma_1 : (d, d, \mathbf{id}, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$.
The *indeterminate inserted–in from* $\mathcal{S}_0$ *to* $\mathcal{S}_1$, $(\otimes \int c_1) : s \rightarrow d$, is the (possibly indeterminate) function:

$$\forall\, t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) :\ \mathcal{B}((\otimes \int c_1)\ t) := \{(\mathsf{TR}\ t')^{\mathcal{A}} | t' \in [t]\}.$$

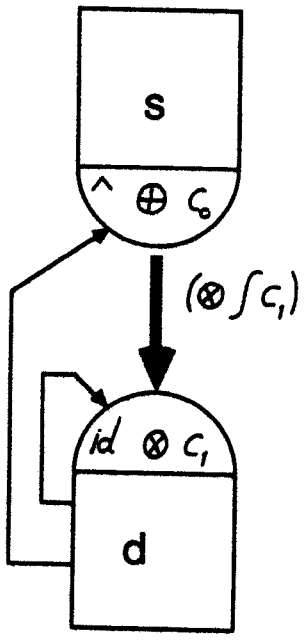

Figure 3.8: Indeterminate inserted–in.

Proposition 42 (*determinacy of indeterminate inserted–in*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (d, d, \mathbf{id}, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$. Let $\mathcal{S}_0$ be manageable and $\mathcal{S}_0 \preceq \mathcal{S}_1$. The indeterminate inserted–in $(\otimes \int c_1)$ is determinate, and $(\otimes \int c_1) = \otimes/$.

Proof: By proposition 40

$$\{(\mathsf{TR}\ t')^{\mathcal{A}} | t' \in [t]\} \subseteq \{t_0{}^{\mathcal{A}} | t_0{}^{\mathcal{A}} = (\mathsf{TR}\ t)^{\mathcal{A}}\} = \{(\mathsf{TR}\ t)^{\mathcal{A}}\}.$$

Since $(\mathsf{TR}\ t)^{\mathcal{A}} \in \{(\mathsf{TR}\ t')^{\mathcal{A}} | t' \in [t]\}$, it follows that

$$\{(\mathsf{TR}\ t')^{\mathcal{A}} | t' \in [t]\} = \{(\mathsf{TR}\ t)^{\mathcal{A}}\},$$

and hence there is only one choice in definition 41: $((\otimes \int c_1)t)^{\mathcal{A}} = (\mathsf{TR}\ t)^{\mathcal{A}}$. One may conclude that $(\otimes \int c_1)$ satisfies the equations of $(\otimes/)$ in proposition 33. □

3.4 General divide–and–rule: Integrates

A curious fact about the present morphic programming framework is that natural operations like the sum of a set of numbers cannot be conveniently expressed. Therefore, they are excluded from the transformation game, which is a pity. The problem is partly overcome in Bird's theory of lists ([Bir86b]) by the introduction of the right– and left–reduce.

$$(\oplus \not/ e)[a_0, a_1, \ldots, a_{n-1}] = (a_0 \oplus (a_1 \oplus (\ldots (a_{n-1} \oplus e)\ldots))) \text{ (right–reduce)}$$

$$(\oplus \not\!/ e)[a_0, a_1, \ldots, a_{n-1}] = (\ldots((e \oplus a_0) \oplus a_1)\ldots \oplus a_{n-1}) \text{ (left–reduce)}$$

However, this solution depends on the unique decomposition of lists into a head and tail (lead and last, respectively). It does not work for other structures.

The definition of indeterminate inserted–in suggests a more general approach: restriction of the congruence class in the set comprehension to achieve determinacy. Homomorphisms reflect the well–known divide–and–rule paradigm to a limited extent. They are a special case of this program scheme in that the divide strategy is not specified. What we intend to do

is this: Restrict the congruence class by specification of the divide strategy. For example, the sum of a set is defined adequately by

$$\begin{array}{lcl} sum\,\emptyset & = & 0 \\ sum\,\{a\} & = & a \\ sum\,x \cup y & = & (sum\,s) + (sum\,t) \\ & & \textbf{where } s \cup t = x \cup y \\ & & \textbf{and } s \cap t = \emptyset. \end{array}$$

3.4.1 Specification of the divide strategy

Definition 43 (*divide function*)
Let $\Sigma : (d, s, \hat{}, \oplus, c)$, $\mathcal{S} : \Sigma$. $\varepsilon : s \to s \times s$ is a *divide function on* $\mathcal{S}$ if $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma)$ either $\mathcal{A} \models (\pi_1 \varepsilon t) \oplus (\pi_2 \varepsilon t) = t$, or $\mathcal{A} \models (\varepsilon\ t) = \bot_s$.

In reasoning about divide strategies, only the breadth of the splitting function is important. Therefore, we do not require that divide functions are total.

Given a divide function ε, one may wonder what terms are taken apart down to atomic level by ε. These expressions are characterized by the parsing predicate defined hereafter.

Definition 44 (*parsing predicate*)
Let $\Sigma : (d, s, \hat{}, \oplus, c)$, $\mathcal{S} : \Sigma$, and ε a divide function on $\mathcal{S}$. The *parsing predicate of* ε, p_ε, is the least predicate on $\mathsf{vterms}(\mathsf{T}, \Sigma)$ satisfying:

(i) $p_\varepsilon c$ holds.

(ii) $\forall a \in \mathsf{grterms}(\mathsf{T}_{sign})_d : p_\varepsilon(\hat{}a)$ holds.

(iii) $\forall t_0, t_1 \in \mathsf{vterms}(\mathsf{T}, \Sigma) : p_\varepsilon(t_0 \oplus t_1)$ holds if both $(p_\varepsilon t_0)$ and $(p_\varepsilon t_1)$ hold and $(\varepsilon(t_0 \oplus t_1)) \sqsupseteq (t_0, t_1)$.

As noted before, the inserted-in (+/) is not defined on sets because union ($\cup$) satisfies idempotency, while addition (+) doesn't. The parsing predicate specifies how the domain of an operator may be restricted such that 'undesirable equalities' (like idempotency above) are eliminated. One may use the predicate to reduce the algebraic richness of an operator.

Example 45
Let

$$\Sigma : (d, dset, \{\_\}, \cup, \emptyset),\ \mathcal{S} :\ \Sigma.$$

Let $\varepsilon : dset \times dset \rightarrow dset$ be defined by:

$$(\mathcal{B}\ \varepsilon\ x) := \{(u, v) | u \cup^{\mathcal{A}} v = x, u \cap^{\mathcal{A}} v = \emptyset^{\mathcal{A}}\}.$$

This specifies a divide function on $\mathcal{S}$. The predicate p_ε may be used to turn $\cup$ into a partial operator $\cup_\varepsilon$, that only accepts disjoint arguments. The original operator $\cup$ is idempotent, whereas $\cup_\varepsilon$ is not.

Definition 46 (*parsed congruence class*)
Let $\Sigma :\ (d, s, \hat{}, \oplus, c)$, $\mathcal{S} :\ \Sigma$. Let $t \in \mathsf{vterms}(\mathsf{T}, \Sigma)$. The *$\varepsilon$-parsed congruence class of t*, $[t]^\Sigma_\varepsilon$, is the set of terms:

$$[t]^\Sigma_\varepsilon := \{t' | t' \in [t],\ p_\varepsilon t'\}.$$

Convention 47 (*omission of structure signature in parsed congruence class*)
If the structure signature Σ is clear from the context (usually quantification over a set of terms) it may be omitted in $[t]^\Sigma_\varepsilon$:

$$[t]_\varepsilon := [t]^\Sigma_\varepsilon.$$

The parsed congruence class reflects the restriction–by–divide principle. It will be convenient if there is always a correct way of partitioning the argument. This amounts to the requirement that there is some way to terminate the evaluation of a divide–and–rule algorithm.

Definition 48 (*terminable divide function*)
Let $\Sigma :\ (d, s, \hat{}, \oplus, c)$, $\mathcal{S} :\ \Sigma$. ε is called a *terminable divide function* if

$$\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma) : [t]_\varepsilon \neq \emptyset.$$

There is another, more abstract interpretation of 'terminable'. Let everything be as in the definition above. In example 45, it was indicated how

the parsing predicate p_ϵ induces a partial version ($\oplus_\epsilon$) of the operator ($\oplus$). This gives rise to the structure

$$\Sigma_\epsilon : (d, s, \hat{}, \oplus_\epsilon, c),\ \mathcal{S}_\epsilon : \Sigma_\epsilon.$$

The divide function ε is terminable iff $\mathcal{S}_\epsilon$ does not contain more junk than $\mathcal{S}$ ($s^{\mathcal{A}}|_{\Sigma_\epsilon} = s^{\mathcal{A}}|_\Sigma$). We will not try to render this conjecture more explicit here.

Of course, the completely free divide strategy employed by homomorphisms satisfies all these requirements. It is specified in squiggles as

$$\sigma z := \square/(\lambda(x, y).x \oplus y = z) \triangleleft s \times s,$$

where $\oplus$ is the binary operator in the structure signature. For any predicate p, $p\triangleleft$ is a 'filter' that removes all elements not satisfying the predicate from its argument. Note that the indeterminate choice is inserted between a possibly infinite number of arguments. A more conventional specification of the breadth of $\sigma\ t$ (with t a visible term of appropriate sort) reads:

$$\mathcal{B}(\sigma\ t) := \{(x, y) \in s^{\mathcal{A}} \times s^{\mathcal{A}} | x \oplus^{\mathcal{A}} y = t^{\mathcal{A}}\}.$$

Proposition 49 (*broadest divide function*)
Let $\Sigma : (d, s, \hat{}, \oplus, c)$, $\mathcal{S} : \Sigma$. Let

$$\sigma z := \square/(\lambda(x, y).x \oplus y = z) \triangleleft s \times s.$$

(i) σ is a divide function on $\mathcal{S}$.

(ii) $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma) : [t]_\sigma = [t]$.

(iii) σ is terminable.

Proof:

(i) Trivial.

(ii) $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma) : p_\sigma t$, by induction over t.

(iii) $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma) : t \in [t]_\sigma$: see (ii).

□

3.4.2 Integrates

A general construct to denote divide–and–rule algorithms is given below. To emphasize the connection to "formal integration" as introduced by M. Sharir in [Sha82], it is called an "integrate". Integrates are powerful expressions, encompassing peculiar cases like the indeterminate inserted–in from the preceding section. They closely resemble the program schemes proposed by D.R. Smith in [Smi85,Smi87a,Smi87b]. Again, it is conjectured without proof that the construct is monotonic with respect to the refinement relation. A graphical illustration of the definition is given in figure 3.9.

Definition 50 (*integrate*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (d, u, f, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$.
Let ε be a terminable divide function on $\mathcal{S}_0$. The *ε–integrate from $\mathcal{S}_0$ to $\mathcal{S}_1$*, $\underset{c_1}{\otimes\!\!\!\!\int} f\ \partial\varepsilon : s \to u$, is a (possibly indeterminate) function: $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0)$

$$\mathcal{B}(\underset{c_1}{\otimes\!\!\!\!\int} f\ \partial\varepsilon\ \ t) := \{(\mathsf{TR}\ t')^{\mathcal{A}} | t' \in [t]_\varepsilon\}.$$

Example 51
Let $\mathcal{S}_0$ be the structure of sets of natural numbers. Let $\mathcal{S}_1$ be the structure of natural numbers, with operator $\times$ and constant 1. The product of a set x is:

$$\prod_{a \in x} a = \underset{1}{\times\!\!\!\!\int}\ \mathrm{id}\ \partial\tau\ \ x$$

where for any term $t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0)$,

$$\mathcal{B}(\tau\ t) := \{(u, v) \in \mathsf{natset}^{\mathcal{A}} \times \mathsf{natset}^{\mathcal{A}} | u \cup^{\mathcal{A}} v = t^{\mathcal{A}}, u \cap^{\mathcal{A}} v = \emptyset^{\mathcal{A}}\}.$$

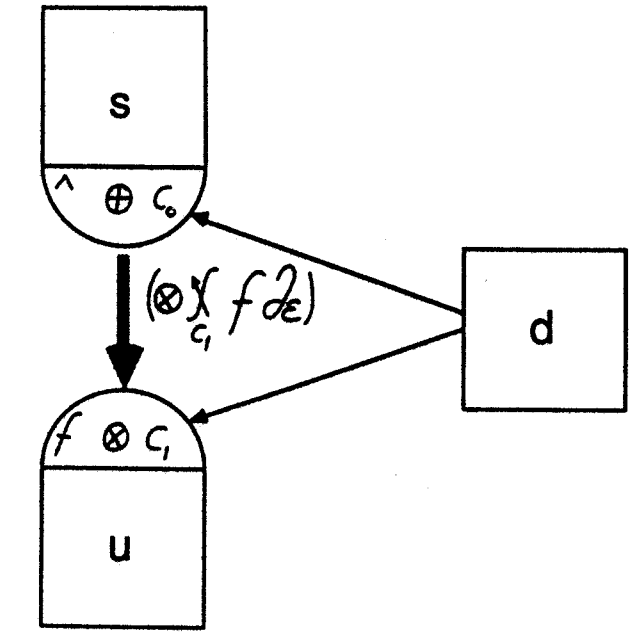

Figure 3.9: Integrate.

Example 52
Let $\mathcal{S}_0$ be the structure of sets of natural numbers. Let $\mathcal{S}_1$ be the structure of lists of natural numbers. Selection sort is specified by:

$$+\!\!+\underset{[]}{\curlyvee}\ [\_]\ \partial\varepsilon$$

where $\varepsilon\ x := (\{\downarrow/x\}, x - \{\downarrow/x\})$. The downward arrow, $\downarrow$ is a binary operator, yielding the minimum of its arguments. It induces $(\downarrow/)$, an inserted–in from sets to naturals, specifying the least element of a set. Note that ε is partial, it is not defined on the empty set.

The most important issue in using integrates is whether they are determinate or not. In the examples above, the details were ignored. The following proposition states a sufficient and necessary condition implying determinacy.

Proposition 53 (*determinacy of an integrate*)
Let $\Sigma_0\ :\ (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0\ :\ \Sigma_0$, $\Sigma_1\ :\ (d, u, f, \otimes, c_1)$, $\mathcal{S}_1\ :\ \Sigma_1$.
The integrate $\otimes\underset{c_1}{\curlyvee}\ f\ \partial\varepsilon$ is determinate iff

$$\forall t \in \mathsf{vterms}(\mathsf{T},\ \Sigma_0) : p_\varepsilon\ t \Rightarrow \{\mathsf{TR}\ t' | t' \in [t]_\varepsilon\} \subseteq [\mathsf{TR}\ t].$$

Proof:

($\Rightarrow$) $\otimes\underset{c_1}{\curlyvee}\ f\ \partial\varepsilon$ is determinate. Let $t \in \mathsf{vterms}(\mathsf{T},\ \Sigma_0)$ be an arbitrary term such that $p_\varepsilon\ t$ holds. By determinacy, there is only one choice possible in definition 50, hence: $\{(\mathsf{TR}\ t')^{\mathcal{A}} | t' \in [t]_\varepsilon\} = \{(\mathsf{TR}\ t)^{\mathcal{A}}\}$, i.e.

$$\forall t' \in [t]_\varepsilon : (\mathsf{TR}\ t')^{\mathcal{A}} = (\mathsf{TR}\ t)^{\mathcal{A}}.$$

The inclusion follows.

($\Leftarrow$) $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : p_\varepsilon\, t \Rightarrow \{\mathsf{TR}\, t' | t' \in [t]_\varepsilon\} \subseteq [\mathsf{TR}\, t]$.
Let $t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0)$ be an arbitrary term. Since ε is terminable: $\exists t_0 \in [t]_\varepsilon$. By definition 46: $(p_\varepsilon\, t_0)$ and hence

$$\{\mathsf{TR}\, t' | t' \in [t_0]_\varepsilon\} \subseteq [\mathsf{TR}\, t_0],$$

i.e. $\{(\mathsf{TR}\, t')^{\mathcal{A}} | t' \in [t_0]_\varepsilon\} = \{(\mathsf{TR}\, t_0)^{\mathcal{A}}\}$, hence the integrate is determinate.

□

Of course, many integrates are homomorphisms. It is important to identify these cases, for this enables us to carry results on integrates over to homomorphisms. First, some basic facts on the refinement of integrates are needed.

The divide function controls the amount of indeterminacy in an integrate. If the divide function is "more defined", then the integrate will not become "less defined". This is expressed in the following proposition.

Proposition 54 (*specialization*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (d, u, f, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$.
Let ε and ϑ be terminable divide functions on $\mathcal{S}_0$, such that $\varepsilon \sqsupseteq \vartheta$. Then

$$\otimes\!\!\!\!\!\!\int_{c_1} f\ \partial\varepsilon \ \sqsupseteq \otimes\!\!\!\!\!\!\int_{c_1} f\ \partial\vartheta\ .$$

Proof: $\varepsilon \sqsupseteq \vartheta$, i.e. $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : \mathcal{B}(\varepsilon\, t) \supseteq \mathcal{B}(\vartheta\, t)$. By induction on t: $p_\vartheta t \Rightarrow p_\varepsilon t$, hence $[t]_\varepsilon \supseteq [t]_\vartheta$. By the definition of integrates (definition 50) $\mathcal{B}(\otimes\!\!\!\!\!\!\int_{c_1} f\ \partial\varepsilon\ t) \supseteq \mathcal{B}(\otimes\!\!\!\!\!\!\int_{c_1} f\ \partial\vartheta\ t)$, i.e. $(\otimes\!\!\!\!\!\!\int_{c_1} f\ \partial\varepsilon\ t) \sqsupseteq (\otimes\!\!\!\!\!\!\int_{c_1} f\ \partial\vartheta\ t)$. □

If an integrate is already determinate, then refinement of the divide function does not alter its semantics. This fact is known as the 'specialization lemma' in Bird's theory of lists [Bir86b]. A practical application might be the implementation of an integrate as a loop (see e.g. [BK80]).

Proposition 55 (*determinate specialization*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (d, u, f, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$. Let ε and ϑ be terminable divide functions on $\mathcal{S}_0$, such that $\varepsilon \sqsupseteq \vartheta$. If $\underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\varepsilon$ is determinate then

$$\underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\varepsilon = \underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\vartheta\ .$$

Proof: Let $\underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\varepsilon$ be determinate. $\underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\varepsilon \sqsupseteq \underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\vartheta$ implies

$$\underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\varepsilon = \underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\vartheta$$

or $(\underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\vartheta\ t)$ is undefined for some $t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0)$. The latter case is in contradiction with the assumption that functions are total (chapter 2). Hence $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : \underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\varepsilon\ t = \underset{c_1}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\vartheta\ t$. □

Intuitively, an integrate is a homomorphism if certain requirements on richness and manageability of structures are met. The next version expresses a strong version of the fact. It is the counterpart of the homomorphism lemma from [Mee86] mentioned before. Figure 3.10 gives an impression of the participating views.

Theorem 56 (*integrate and morphic programming homomorphism*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (e, u, \tilde{}, \odot, c_1)$, $\mathcal{S}_1 : \Sigma_1$, $\Sigma_2 : (d, e, f, \otimes, c_2)$, $\mathcal{S}_2 : \Sigma_2$. If

(i) $\mathcal{S}_0$ is manageable

(ii) $\mathcal{S}_1$ is manageable

(iii) $\mathcal{S}_0 \preceq \mathcal{S}_1 \preceq \mathcal{S}_2$

then

$$\underset{c_2}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\varepsilon = (\otimes/)(f*),$$

where $\underset{c_2}{\otimes\!\!\!\!\!\!\curlywedge} f\ \partial\varepsilon$ the ε–integrate from $\mathcal{S}_0$ to $\mathcal{S}_2$, $(f*)$ the apply–to–all from $\mathcal{S}_0$ to $\mathcal{S}_1$ and $\otimes/$ the inserted–in from $\mathcal{S}_1$ to $\mathcal{S}_2$.

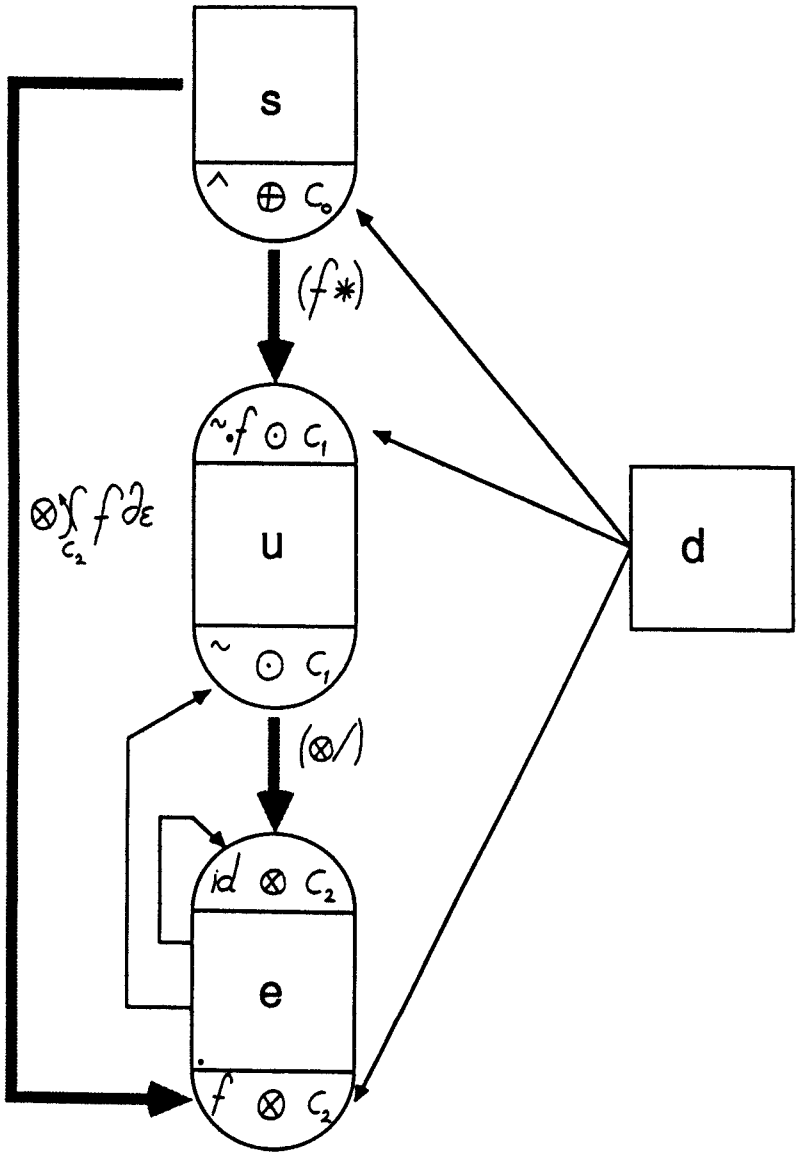

Figure 3.10: Theorem 56.

Proof: Let (i)–(iii) hold. By (i) and (iii), proposition 40 is applicable:

$$\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : \{\mathsf{TR}_{\Sigma_0 \to \Sigma_2} t' | t' \in [t]\} \subseteq [\mathsf{TR}_{\Sigma_0 \to \Sigma_2} t].$$

Hence by proposition 53, $\otimes\!\!\!\!\!\!\int\limits_{c_2} f\ \partial\varepsilon$ is determinate. Proposition 55 yields:

$$\otimes\!\!\!\!\!\!\int\limits_{c_2} f\ \partial\varepsilon\ = \otimes\!\!\!\!\!\!\int\limits_{c_2} f\ \partial\sigma\ ,$$

σ being the broadest divide function from proposition 49. By determinacy and the fact that $t \in [t]_\sigma$ for any $t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0)$, unfolding of definition 50 leads to:

$$\begin{aligned}
\otimes\!\!\!\!\!\!\int\limits_{c_2} f\ \partial\sigma\ c_0 &= c_2 \\
\otimes\!\!\!\!\!\!\int\limits_{c_2} f\ \partial\sigma\ \hat{a} &= fa \\
\otimes\!\!\!\!\!\!\int\limits_{c_2} f\ \partial\sigma\ x \oplus y &= (\otimes\!\!\!\!\!\!\int\limits_{c_2} f\ \partial\sigma\ x) \otimes (\otimes\!\!\!\!\!\!\int\limits_{c_2} f\ \partial\sigma\ y).
\end{aligned}$$

From propositions 33 and 35, it is known that $(\otimes/)(f*)$ is the unique function satisfying these equations, hence:

$$(\otimes/)(f*) = \otimes\!\!\!\!\!\!\int\limits_{c_2} f\ \partial\sigma\ = \otimes\!\!\!\!\!\!\int\limits_{c_2} f\ \partial\varepsilon\ .$$

□

Definition 57 (*homomorphism condition*)
If three structures, $\mathcal{S}_0$, $\mathcal{S}_1$ and $\mathcal{S}_2$ satisfy the conditions (i)–(iii) in theorem 56 they are said to satisfy the *homomorphism condition.*

3.4.3 Composition of integrates

An interesting question concerns the composition of two integrates. When is it possible to merge them into a single one? To solve this problem, some extra theory on structures and terms is needed.

Note that under the present approach, a single carrier may have multiple structures assigned to it. A special case occurs when only the embedding and the constant are differing between two views on the same sort. In such

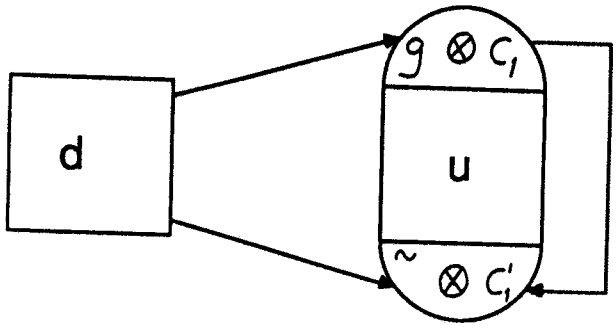

Figure 3.11: Conversion.

cases, it is often possible to convert all terms over the first signature into terms over the second, preserving their semantics. This process is called a conversion from one signature to another. The graphical segment denoting a conversion is a dotted arrow (figure 3.11). It indicates a change of view without losing sight of values.

Definition 58 (*convertible*)
Let $\Sigma_1 : (d, u, g, \otimes, c_1)$, $\Sigma_1' : (d, u, \tilde{}, \otimes, c_1')$. Σ_1 *is convertible to* Σ_1' if

(i) $\exists t_0' \in \mathsf{vterms}(\mathsf{T}, \Sigma_1') : \mathsf{T}_{laws} \vdash t_0' = c_1$

(ii) $\forall a \in \mathsf{grterms}(\mathsf{T}_{sign})_d : \exists t_1' \in \mathsf{vterms}(\mathsf{T}, \Sigma_1') : \mathsf{T}_{laws} \vdash t_1' = g\ a.$

Definition 59 (*conversion*)
Let $\Sigma_1 : (d, u, g, \otimes, c_1)$, $\Sigma_1' : (d, u, \tilde{}, \otimes, c_1')$, and let Σ_1 be convertible to Σ_1'.
If $\mathsf{C} : \mathsf{vterms}(\mathsf{T}, \Sigma_1) \rightarrow \mathsf{vterms}(\mathsf{T}, \Sigma_1')$, satisfies:

(i) $\mathsf{T}_{laws} \vdash \mathsf{C}\, c_1 = c_1$

(ii) $\forall a \in \mathsf{grterms}(\mathsf{T}_{sign})_d : \mathsf{T}_{laws} \vdash \mathsf{C}(g\ a) = (g\ a)$

(iii) $\forall t_0, t_1 \in \mathsf{vterms}(\mathsf{T}, \Sigma_1) : \mathsf{T}_{laws} \vdash \mathsf{C}(t_0 \otimes t_1) = (\mathsf{C}t_0) \otimes (\mathsf{C}t_1)$

then C is called a *conversion from* Σ_1 *to* Σ_1'.

Proposition 60 (*conversion is conservative*)
Let everything be as in definition 59.

$$\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_1) : \mathsf{T}_{laws} \vdash (\mathsf{C}t){=}t.$$

Proof: Induction on t (fact 16). □

Conversion is nothing new. It is common practice in morphic programming to adopt a standard view on a sort. No matter by what term translation one arrives there, values are represented as terms over this standard signature. The implicit conversion goes unmentioned.

Example 61
As before, let $[\alpha]$ denote the sort of lists with elements of sort α.

$$\begin{array}{lll} \Sigma_0 : ([\mathsf{int}], [[\mathsf{int}]], [\_]_{[\mathsf{int}]}, +\!\!+_{[[\mathsf{int}]]}, []_{[[\mathsf{int}]]}), & \mathcal{S}_0 : \Sigma_0 \\ \Sigma_1 : ([\mathsf{int}], [\mathsf{int}], \mathbf{id}_{[\mathsf{int}]}, +\!\!+_{[\mathsf{int}]}, []_{[\mathsf{int}]}), & \mathcal{S}_1 : \Sigma_1 \\ \Sigma_2 : (\mathsf{int}, [\mathsf{int}], [\_]_{\mathsf{int}}, +\!\!+_{[\mathsf{int}]}, []_{[\mathsf{int}]}), & \mathcal{S}_2 : \Sigma_2 \end{array}$$

The inserted–in $+\!\!+_{[\mathsf{int}]}$/ from $\mathcal{S}_0$ to $\mathcal{S}_2$ is defined by a term translation from Σ_0 to Σ_1. However, the result is considered as an expression from $\mathsf{vterms}(\mathsf{T}, \Sigma_2)$. In examples 4, 5, 6 other views on [int](= iseq) are listed.

Proposition 62 (*no junk and convertibility*)
Let $\Sigma_1' : (d, u, \tilde{}, \otimes, c_1')$ be such that $u^{\mathcal{A}}|_{\Sigma_1'} = u^{\mathcal{A}}$. Then for any $\Sigma_1 : (d, u, g, \otimes, c_1)$, Σ_1 is convertible to Σ_1'.

Proof: Immediate from definitions 11 and 58. □

The following theorem is central to this paper. It states that under extremely general conditions, two divide–and–rule algorithms may be merged into a single one. This is a remarkable result, and one wonders whether similar laws hold for programs on more complex structures. Because of its importance, the proof of this theorem is elaborated in detail. The views involved in the theorem are displayed in figure 3.12.

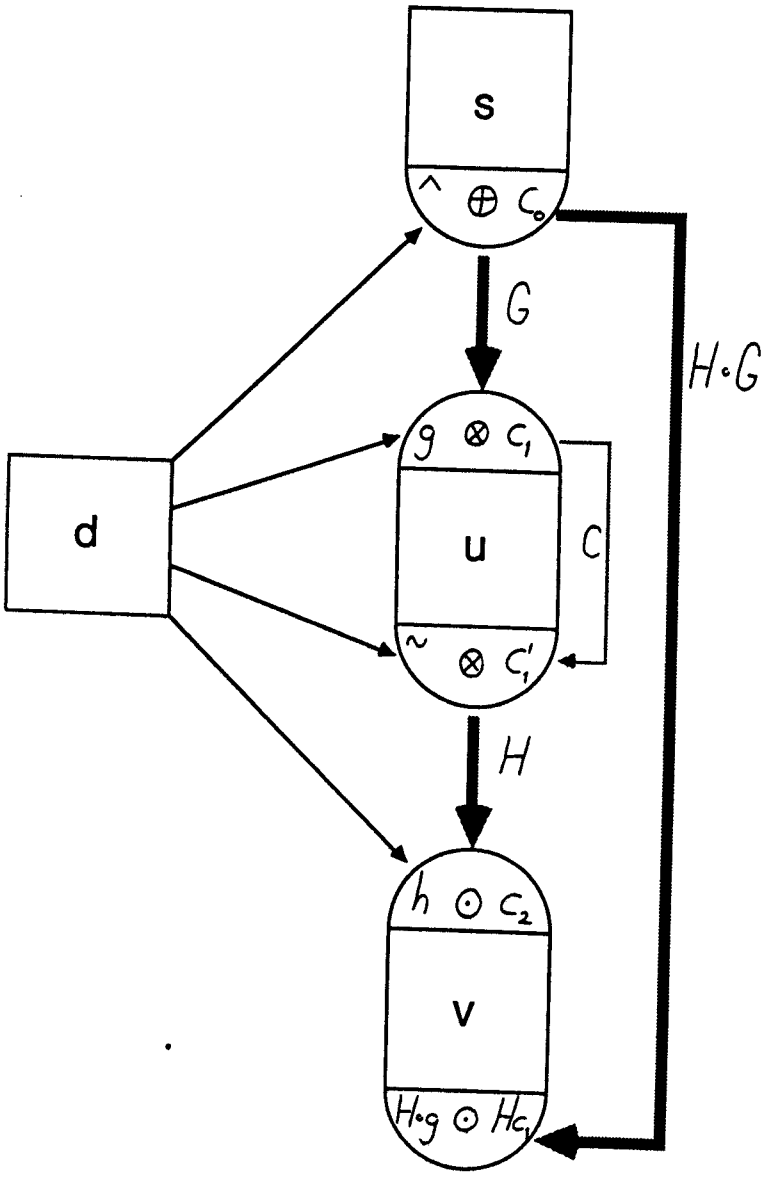

Figure 3.12: Theorem 63.

Theorem 63 (*integrate composition*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (d, u, g, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$, $\Sigma_1' : (d, u, \tilde{}, \otimes, c_1')$, $\mathcal{S}'_1 : \Sigma_1'$, $\Sigma_2 : (d, v, h, \odot, c_2)$, $\mathcal{S}_2 : \Sigma_2$, with Σ_1 convertible to Σ_1' by conversion C.
Let G be the ε–integrate from $\mathcal{S}_0$ to $\mathcal{S}_1$, and H be the ϑ–integrate from $\mathcal{S}'_1$ to $\mathcal{S}_2$.
Let $\Sigma_3 : (d, v, f, \odot, c_3)$, $\mathcal{S}_3 : \Sigma_3$, such that $\mathcal{A}$ satisfies $f = H \circ g$ and $c_3 = Hc_1$.
If

(i) H is determinate

(ii) $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : p_\varepsilon t \Rightarrow p_\vartheta(\mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1}\ t)$

then $H \circ G$ is the ε–integrate from $\mathcal{S}_0$ to $\mathcal{S}_3$.

Proof: Let (i) and (ii) hold. Let $t_0' := (\mathsf{C}\ c_1)$ and $\forall a \in \mathsf{grterms}(\mathsf{T}_{sign})_d$: $t_1'[a] := \mathsf{C}(g\ a)$.

Lemma 64 (*translation composition*)

$$\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : (\mathsf{TR}_{\Sigma_1' \to \Sigma_2}\ \mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1}\ t)^{\mathcal{A}} = (\mathsf{TR}_{\Sigma_0 \to \Sigma_3}\ t)^{\mathcal{A}}$$

Proof: Induction on t.
$t = c_0$

$$\begin{array}{lcl} (\mathsf{TR}_{\Sigma_1' \to \Sigma_2}\ \mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1}\ t)^{\mathcal{A}} & = & \text{(def. 26)} \\ (\mathsf{TR}_{\Sigma_1' \to \Sigma_2}\ \mathsf{C}\ c_1)^{\mathcal{A}} & = & \text{(def. 59)} \\ (\mathsf{TR}_{\Sigma_1' \to \Sigma_2}\ t_0')^{\mathcal{A}} & & \end{array}$$

By definition 50 we have

$$\mathcal{B}(H\ t_0') = \{(\mathsf{TR}_{\Sigma_1' \to \Sigma_2}\ t_0'')^{\mathcal{A}} | t_0'' \in [t_0]_\vartheta\}.$$

$(p_\epsilon\ c_0)$ holds by definition of the parsing predicate (definition 44). Therefore, according to assumption (ii), $(p_\epsilon\ t'_0)$ holds, hence

$$(\mathsf{TR}_{\Sigma'_1 \to \Sigma_2}\ t'_0)^{\mathcal{A}} \in \{(\mathsf{TR}_{\Sigma'_1 \to \Sigma_2}\ t''_0)^{\mathcal{A}} | t''_0 \in [t_0]_\vartheta\}.$$

Assumption (i), together with the assumption that functions are total (section 2.3) yields that $\mathcal{B}(H\ t'_0)$ is a singleton set. Hence

$$(\mathsf{TR}_{\Sigma'_1 \to \Sigma_2}\ t'_0)^{\mathcal{A}} = (H\ t'_0)^{\mathcal{A}}.$$

The derivation proceeds:

$$\begin{array}{lcl}
(H\ t'_0)^{\mathcal{A}} & = & \text{(def. interpretation)} \\
H^{\mathcal{A}}\ t'^{\mathcal{A}}_0 & = & \text{(given)} \\
H^{\mathcal{A}}\ (\mathsf{C}\ c_1)^{\mathcal{A}} & = & \text{(prop. 60)} \\
H^{\mathcal{A}}\ c_1{}^{\mathcal{A}} & = & \text{(def. interpretation)} \\
(H\ c_1)^{\mathcal{A}} & = & \text{(given)} \\
c_3{}^{\mathcal{A}} & = & \text{(def. 26)} \\
(\mathsf{TR}_{\Sigma_0 \to \Sigma_3}\ c_0)^{\mathcal{A}} & = & \\
(\mathsf{TR}_{\Sigma_0 \to \Sigma_3}\ t)^{\mathcal{A}} & & \text{(q.e.d.)}
\end{array}$$

$t = \hat{}a$

$$\begin{array}{lcl}
(\mathsf{TR}_{\Sigma'_1 \to \Sigma_2}\ \mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1}\ t)^{\mathcal{A}} & = & \text{(def. 26)} \\
(\mathsf{TR}_{\Sigma'_1 \to \Sigma_2}\ \mathsf{C}\ (g\ a))^{\mathcal{A}} & = & \text{(def. 59)} \\
(\mathsf{TR}_{\Sigma'_1 \to \Sigma_2}\ t'_1[a])^{\mathcal{A}} & = & \text{(as above by (i)\&(ii))} \\
(H\ t'_1[a])^{\mathcal{A}} & = & \text{(def. interpretation)} \\
H^{\mathcal{A}}(t'_1[a])^{\mathcal{A}} & = & \text{(given)} \\
H^{\mathcal{A}}(\mathsf{C}(g\ a))^{\mathcal{A}} & = & \text{(prop. 60)} \\
H^{\mathcal{A}} g^{\mathcal{A}} a^{\mathcal{A}} & = & \text{(def. interpretation)} \\
(H \circ g)^{\mathcal{A}} a^{\mathcal{A}} & = & \text{(given)} \\
f^{\mathcal{A}} a^{\mathcal{A}} & = & \text{(def. interpretation)} \\
(f\ a)^{\mathcal{A}} & = & \text{(def. 26)} \\
(\mathsf{TR}_{\Sigma_0 \to \Sigma_3}\ \widehat{a})^{\mathcal{A}} & = & \\
(\mathsf{TR}_{\Sigma_0 \to \Sigma_3}\ t)^{\mathcal{A}} & & \text{(q.e.d.)}
\end{array}$$

$t = t_0 \oplus t_1$ **Induction Hypothesis:** For $i = 0, 1$:

$$(\mathsf{TR}_{\Sigma_1' \to \Sigma_2} \mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1} t_i)^{\mathcal{A}} = (\mathsf{TR}_{\Sigma_0 \to \Sigma_3} t_i)^{\mathcal{A}}$$

$$\begin{array}{lcl}
(\mathsf{TR}_{\Sigma_1' \to \Sigma_2} \mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1} t)^{\mathcal{A}} & = & \text{(def. 26)} \\
(\mathsf{TR}_{\Sigma_1' \to \Sigma_2} \mathsf{C}\ ((\mathsf{TR}_{\Sigma_0 \to \Sigma_1} t_0) \otimes (\mathsf{TR}_{\Sigma_0 \to \Sigma_1} t_1)))^{\mathcal{A}} & = & \text{(def. 59)} \\
(\mathsf{TR}_{\Sigma_1' \to \Sigma_2}((\mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1} t_0) \otimes (\mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1} t_1)))^{\mathcal{A}} & = & \text{(def. 26)} \\
((\mathsf{TR}_{\Sigma_1' \to \Sigma_2} \mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1} t_0) \odot (\mathsf{TR}_{\Sigma_1' \to \Sigma_2} \mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1} t_1))^{\mathcal{A}} & = & \text{(def. interpretation)} \\
(\mathsf{TR}_{\Sigma_1' \to \Sigma_2} \mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1} t_0)^{\mathcal{A}} \odot^{\mathcal{A}} (\mathsf{TR}_{\Sigma_1' \to \Sigma_2} \mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1} t_1)^{\mathcal{A}} & = & \text{(induction hypothesis)} \\
(\mathsf{TR}_{\Sigma_0 \to \Sigma_3} t_0)^{\mathcal{A}} \odot^{\mathcal{A}} (\mathsf{TR}_{\Sigma_0 \to \Sigma_3} t_1)^{\mathcal{A}} & = & \text{(def. interpretation)} \\
((\mathsf{TR}_{\Sigma_0 \to \Sigma_3} t_0) \odot (\mathsf{TR}_{\Sigma_0 \to \Sigma_3} t_1))^{\mathcal{A}} & = & \text{(def. 26)} \\
(\mathsf{TR}_{\Sigma_0 \to \Sigma_3} (t_0 \oplus t_1))^{\mathcal{A}} & = & \\
(\mathsf{TR}_{\Sigma_0 \to \Sigma_3} t)^{\mathcal{A}} & & \text{(q.e.d.)}
\end{array}$$

□

Let $t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0)$ be an arbitrary term.

$$\begin{array}{lcl}
\mathcal{B}(H \circ G\ t) & = & (H \text{ determinate }) \\
\{H^{\mathcal{A}} x | x \in \mathcal{B}(Gt)\} & = & \text{(def. 50)} \\
\{H^{\mathcal{A}}(\mathsf{TR}_{\Sigma_0 \to \Sigma_1} t')^{\mathcal{A}} | t' \in [t]_e\} & = & \text{(proposition 60)} \\
\{H^{\mathcal{A}}(\mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1} t')^{\mathcal{A}} | t' \in [t]_e\} & = & \text{(def. 50 and assumption (ii))} \\
\{(\mathsf{TR}_{\Sigma_1' \to \Sigma_2} \mathsf{C}\ \mathsf{TR}_{\Sigma_0 \to \Sigma_1} t')^{\mathcal{A}} | t' \in [t]_e\} & = & \text{(lemma 64)} \\
\{(\mathsf{TR}_{\Sigma_0 \to \Sigma_3} t')^{\mathcal{A}} | t' \in [t]_e\} & &
\end{array}$$

□

Many interesting programming problems may be formulated using right–inverses of homomorphisms ("generalized inverses" in [Bir86a]). For example, if

$$+\!\!+ / \circ part = \mathbf{id}$$

then $(part\ x)$ specifies some partition of the list x. As an immediate consequence of the composition theorem, we have the following general result.

Proposition 65 (*integrate inversion*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$
$\Sigma_1 : (d, u, \tilde{}, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$.
Let G be the ε–integrate from $\mathcal{S}_0$ to $\mathcal{S}_1$, and let H be the ϑ–integrate from $\mathcal{S}_1$ to $\mathcal{S}_0$. If

(i) H is determinate

(ii) $\forall t \in \mathsf{vterms}(\mathsf{T}, \Sigma_0) : p_\varepsilon t \Rightarrow p_\vartheta(\mathsf{C}\ \underset{\Sigma_0 \to \Sigma_1}{\mathsf{TR}}\ t)$

then $H \circ G = \mathbf{id}$.

Example 66
Let $\Sigma_0 : (dset, dsetset, \{\_\}_{ss}, \cup_{ss}, \emptyset_{ss})$, $\mathcal{S}_0 : \Sigma_0$
$\Sigma_1 : (dset, dset, \mathbf{id}, \cup_s, \emptyset_s)$, $\mathcal{S}_1 : \Sigma_1$.
$\cup_s/$ is the inserted–in from $\mathcal{S}_0$ to $\mathcal{S}_1$. According to the integrate inversion proposition,

$$(\cup_s/)(\cup_{ss} \underset{\emptyset_{ss}}{\curlywedge} \{\_\}_{ss}\ \partial\sigma\ ) = \mathbf{id}$$

where σ is the broadest divide function from proposition 49. The integrate

$$\cup_{ss} \underset{\emptyset_{ss}}{\curlywedge} \{\_\}_{ss}\ \partial\sigma\ x$$

specifies some partition of x with possibly overlapping elements.

If one intends to exclude non–disjoint pairs from a partition, the integrate could be specialized to

$$setpart = \cup_{ss} \underset{\emptyset_{ss}}{\curlywedge} \{\_\}_{ss}\ \partial\varepsilon\ ,$$

where

$$\varepsilon\ x = \square/\{(u, v) | u \cup v = x, u \cap v = \emptyset\}.$$

Now this is a remarkable function, for many problems may be formulated in terms of its breadth. The expression

$$\downarrow_\#/\ (all\ p) \triangleleft \mathcal{B}(setpart\ x)$$

specifies a coarsest partition of x with all elements satisfying p. (Properly speaking, the expression above is not a sentence from our language. This

anomaly is ignored.) We return to partition problems in the next chapter.

If H is a homomorphism, the conditions in the integrate composition theorem are always satisfied, which accounts for the neat promotion laws as presented in the morphic programming papers. Here, an intermediate, slightly more general result is proved first. It is illustrated in figure 3.13.

Proposition 67 (*composition of integrate and homomorphism*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (d, u, g, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$, and G the ε–integrate from $\mathcal{S}_0$ to $\mathcal{S}_1$.
Let $\Sigma_1' : (d, u, \tilde{}, \otimes, c_1')$, $\mathcal{S}'_1 : \Sigma_1'$, $\Sigma_2 : (e, v, \bar{}, \ominus, c_2)$, $\mathcal{S}_2 : \Sigma_2$, $\Sigma_3 : (d, e, h, \odot, c_3)$, $\mathcal{S}_3 : \Sigma_3$, be such that $\mathcal{S}'_1$,$\mathcal{S}_2$, $\mathcal{S}_3$ satisfy the homomorphism condition.
Let $(h*)$ be the apply–to–all from $\mathcal{S}'_1$ to $\mathcal{S}_2$, and $(\odot/)$ be the inserted–in from $\mathcal{S}_2$ to $\mathcal{S}_3$.
Let $\Sigma_4 : (d, e, H \circ g, \odot, Hc_1)$, $\mathcal{S}_4 : \Sigma_4$.

$$H \circ G = \bigcirc\!\!\!\!\!\!\int_{\odot/h*c_1} \odot/h * g \; \partial\varepsilon \;,$$

the ε–integrate from $\mathcal{S}_0$ to $\mathcal{S}_4$.

Proof: By theorem 56, H is the σ–integrate from $\mathcal{S}'_1$ to $\mathcal{S}_3$, $\odot/h* = \bigcirc\!\!\!\!\!\!\int_{c_3} h \; \partial\sigma$, σ the broadest divide function from proposition 49. $\odot/h*$ is determinate. By proposition 49, condition (ii) in theorem 63 is satisfied too. Finally, Σ_1 is reducible to Σ_1' by proposition 62. Application of the integrate composition theorem yields the desired result. □

Example 68
Let *sort* : natset → natseq
p : nat → bool.
The conventional structures of sets of naturals and lists of naturals are assumed. It is intended to show by calculation

$$p\triangleleft \; sort = sort \; p\triangleleft.$$

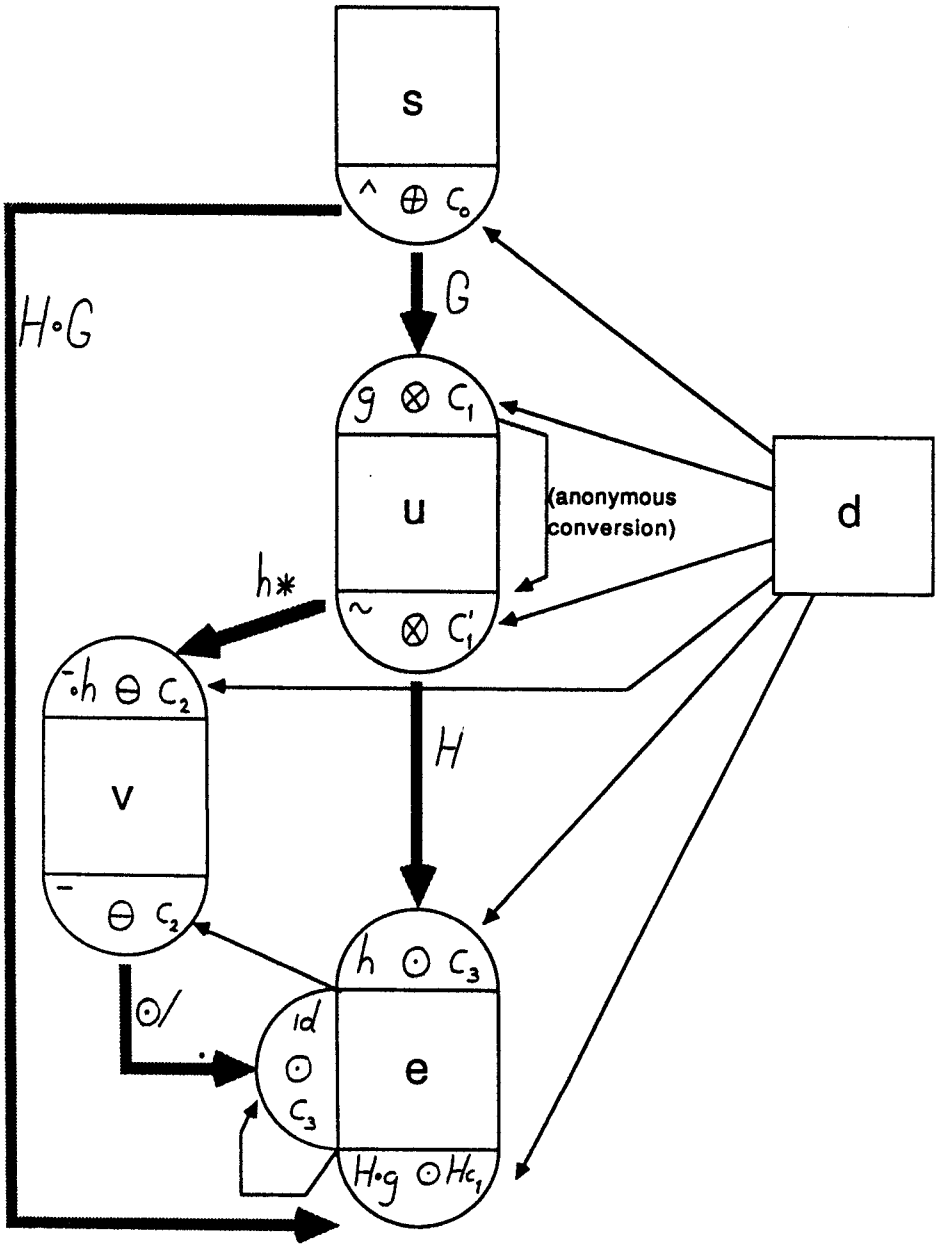

Figure 3.13: Proposition 67.

For brevity, define the following functions

$$\begin{array}{lrcl} f^l_p\ a := & p\ a & \to & [a] \\ \square & \neg p\ a & \to & [\,] \end{array}$$

$$\begin{array}{lrcl} f^s_p\ a := & p\ a & \to & \{a\} \\ \square & \neg p\ a & \to & \emptyset. \end{array}$$

We use the definition of selection sort from example 52.

$$\begin{array}{lcl} p\triangleleft\ sort & = & \text{(example 52)} \\ p\triangleleft\ selsort & = & \text{(unfold)} \\ +\!\!+/\ f^l_p* \ +\!\!+\!\!\!\!\!\!\nwarrow_{[]}\ [\_]\ \partial\varepsilon & = & \text{(prop. 67)} \\ +\!\!+\!\!\!\!\!\!\nwarrow_{[]}\ +\!\!+/f^l_p*\ [\_]\ \partial\varepsilon & = & \\ +\!\!+\!\!\!\!\!\!\nwarrow_{[]}\ f^l_p\ \partial\varepsilon & = & \\ +\!\!+\!\!\!\!\!\!\nwarrow_{[]}\ (+\!\!+\!\!\!\!\!\!\nwarrow_{[]}\ [\_]\ \partial\varepsilon\ f^s_p)\ \partial\varepsilon & = & \text{(theorem 63)} \\ +\!\!+\!\!\!\!\!\!\nwarrow_{[]}\ [\_]\ \partial\varepsilon\ \cup\!\!\!\!\!\!\nwarrow_{\emptyset}\ f^s_p\ \partial\varepsilon & = & \text{(theorem 56)} \\ +\!\!+\!\!\!\!\!\!\nwarrow_{[]}\ [\_]\ \partial\varepsilon\ \cup/f^s_p* & = & \text{(fold)} \\ selsort\ p\triangleleft & = & \\ sort\ p\triangleleft & & \end{array}$$

The following proposition is a concise formulation of the celebrated promotion laws in morphic programming. By careful choice of structures, they emerge from the (admittedly obscuring) flood of squiggles. The result is illustrated in figure 3.14.

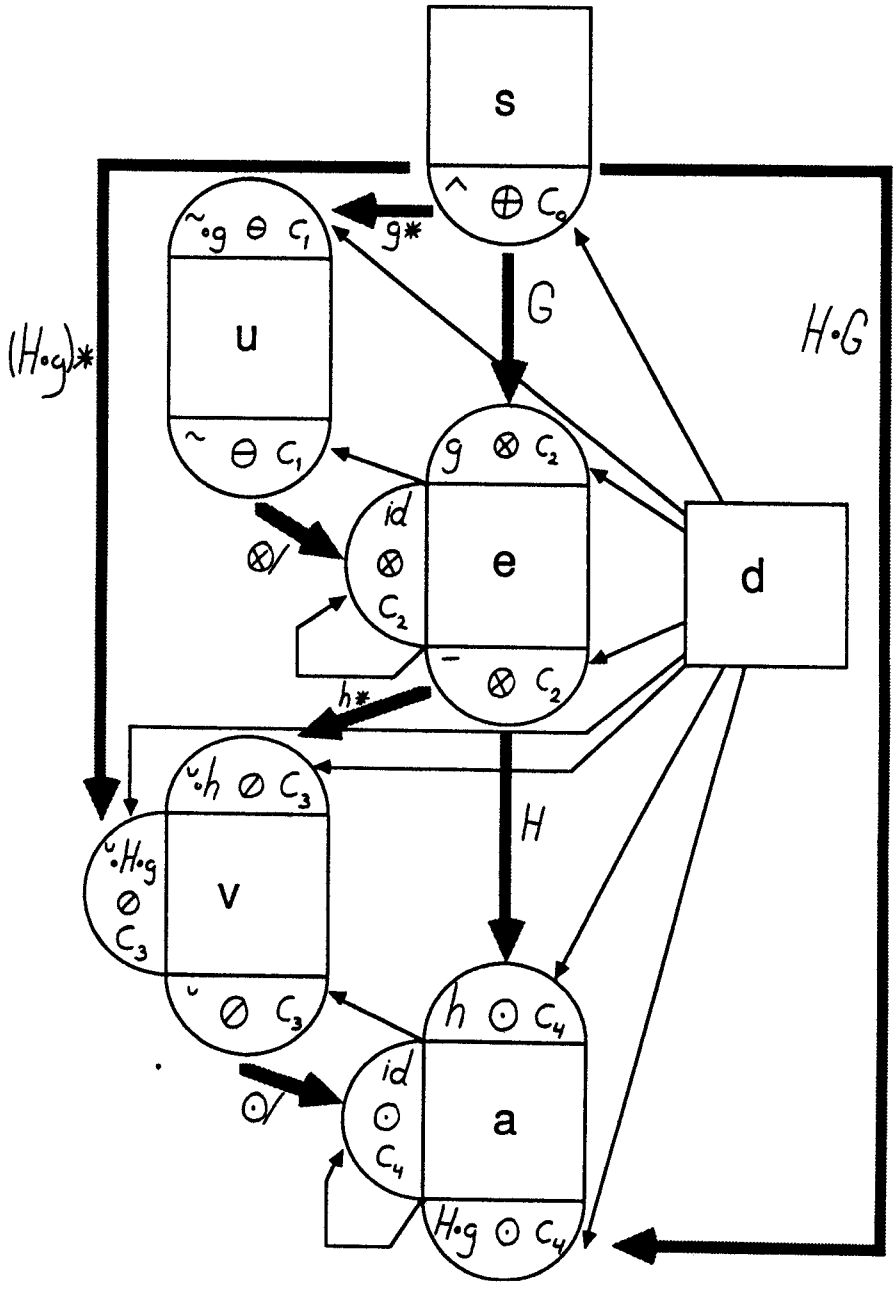

Figure 3.14: Proposition 69.

Proposition 69 (*homomorphism composition*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$, $\Sigma_1 : (e, u, \tilde{}, \ominus, c_1)$, $\mathcal{S}_1 : \Sigma_1$, $\Sigma_2 : (d, e, g, \otimes, c_2)$, $\mathcal{S}_2 : \Sigma_2$, be such that $\mathcal{S}_0$, $\mathcal{S}_1$ and $\mathcal{S}_2$ satisfy the homomorphism condition. Let $G := \otimes/g*$, $(g*)$ the apply–to–all from $\mathcal{S}_0$ to $\mathcal{S}_1$, $\otimes/$ the inserted–in from $\mathcal{S}_1$ to $\mathcal{S}_2$.
Let $\Sigma'_2 : (d, e, \bar{}, \otimes, c_2')$, $\mathcal{S}'_2 : \Sigma'_2$, $\Sigma_3 : (a, v, \check{}, \oslash, c_3)$, $\mathcal{S}_3 : \Sigma_3$, $\Sigma_4 : (d, a, h, \odot, c_4)$, $\mathcal{S}_4 : \Sigma_4$, be such that $\mathcal{S}'_2$, $\mathcal{S}_3$ and $\mathcal{S}_4$ satisfy the homomorphism condition. Let $H := \odot/h*$, $(h*)$ the apply–to–all from $\mathcal{S}'_2$ to $\mathcal{S}_3$, $\odot/$ the inserted–in from $\mathcal{S}_3$ to $\mathcal{S}_4$.
If $c_2 = c_2'$ then

$$H \circ G = \odot/(\odot/h * g)*$$

where $(\odot/h * g)*$ the apply–to–all from $\mathcal{S}_0$ to $\mathcal{S}_3$.

Proof: Let $c_2 = c_2'$. Let $\Sigma_5 : (d, a, H \circ g, \odot, Hc_2)$, $\mathcal{S}_5 : \Sigma_5$. Let σ be the broadest divide function from proposition 49. Theorem 56 yields: $G = \odot\!\!\!\!\!\!\int\limits_{c_2} g \; \partial\sigma$, the σ–integrate from $\mathcal{S}_0$ to $\mathcal{S}_2$. By proposition 67: $H \circ G = \odot\!\!\!\!\!\!\int\limits_{Hc_2} \odot/h * g \; \partial\sigma$, the σ–integrate from $\mathcal{S}_0$ to $\mathcal{S}_5$. Since H is a homomorphism, we have from the assumption $c_2 = c_2'$: $Hc_2 = Hc_2' = c_4$. Therefore, $\$(\mathcal{S}_2) = \$(\mathcal{S}'_2)$ and $\$(\mathcal{S}_4) = \$(\mathcal{S}_5)$. One arrives at (by the homomorphism condition): $\mathcal{S}_0 \preceq \mathcal{S}_2 \preceq \mathcal{S}'_2 \preceq \mathcal{S}_3 \preceq \mathcal{S}_4 \preceq \mathcal{S}_5$. By transitivity of $\preceq$, (prop. 29) $\mathcal{S}_0 \preceq \mathcal{S}_5$. Together with the homomorphism condition on ($\mathcal{S}'_2$, $\mathcal{S}_3$, $\mathcal{S}_4$) this implies that $\mathcal{S}_0$, $\mathcal{S}_3$ and $\mathcal{S}_5$ satisfy the homomorphism condition. Note that $(\odot/)$ is an inserted–in from $\mathcal{S}_3$ to $\mathcal{S}_5$, too. Using theorem 56, $H \circ G = \odot/(\odot/h * g)*$. □

Example 70
Let $[\alpha]$ denote the sort of lists with elements from α. The obvious structure on this sort has signature

$$\begin{aligned} \alpha - \text{list} =< \{\alpha, [\alpha]\}, \{\ & [\_]_\alpha && : \alpha \to [\alpha] \\ & \_ +\!\!+_{[\alpha]} \_ && : [\alpha] \times [\alpha] \to [\alpha] \\ & []_{[\alpha]} && : \to [\alpha] \} > . \end{aligned}$$

Let $(h : \alpha \to \beta)$ be a function. The corresponding apply–to–all is

$$h* : [\alpha] \to [\beta].$$

Note that

$$h* = +\!\!+_{[\beta]}/([\_]_\beta\, h)*, \text{ and}$$
$$+\!\!+_{[\alpha]}/ = +\!\!+_{[\alpha]}/(\mathbf{id}_{[\alpha]}*).$$

From the proposition above (prop. 69), we have

$$h*\ +\!\!+_{[\alpha]}/ = +\!\!+_{[\beta]}/(h*)*.$$

In a similar way, other promotion laws may be derived. Of course, it is more convenient to prove promotion laws directly using the results on apply–to–all and inserted–in. However, the proof by way of proposition 69 reveals the connection to general divide–and–rule algorithms.

Example 71

Given a function f and a predicate p, we intend to compute the number of elements in a set with an f–value that satisfies p. The conventional set– and bag–structures are used:

$$\begin{array}{ll} \textbf{setsign} & : (d, dset, \{\_\}, \cup, \emptyset) \\ \textbf{bagsign} & : (d, dbag, <\_>, \sqcup, <>). \end{array}$$

The problem is specified by

$$H = \#\ p \triangleleft\ f*\ \ bagify$$

where *bagify* turns a set into a bag, with one copy of each element. This function is specified by

$$bagify := \bigsqcup_{<>} <\_>\ \partial\varepsilon\ ,\ \varepsilon\ x := \square/\{(s,t) | s \cup t = s \cap t = \emptyset\}.$$

To ease the calculation of a solution, the following notation is introduced

$$x \triangleleft q \triangleright y := \begin{array}{lll} q & \rightarrow & x \\ \square\ \neg q & \rightarrow & y. \end{array}$$

Another abbreviation is K_1, representing the constant function yielding 1.

$$
\begin{array}{lll}
H & = & \\
\#\, p \triangleleft\, f*\ \textit{bagify} & = & \\
+/K_1*\ \sqcup/(<\_> \triangleleft p\_ \triangleright <>)*\ f*\ \textit{bagify} & = & \text{(prop. 69)} \\
+/(+/K_1*\ (<\_> \triangleleft p\_ \triangleright <>))*\ f*\ \textit{bagify} & = & (g(x \triangleleft q \triangleright y) = \\
 & & (g\ x) \triangleleft q \triangleright (g\ y)) \\
+/(1 \triangleleft p\_ \triangleright 0)*\ f*\ \textit{bagify} & = & (h * g* = (h\, g)*) \\
+/((1 \triangleleft p\_ \triangleright 0)f)*\ \textit{bagify} & = & \\
+/(1 \triangleleft p \circ f\_ \triangleright 0)*\ \textit{bagify} & = & \text{(see above)} \\
+/(1 \triangleleft p \circ f\_ \triangleright 0)*\ \sqcup\!\!\!\!\!\!\int\limits_{<>}^{\kappa} <\_>\ \partial\varepsilon & = & \text{(prop. 67)} \\
+\!\!\!\!\!\!\int\limits_{0}^{\kappa} (+/(1 \triangleleft p \circ f\_ \triangleright 0)* <\_>)\ \partial\varepsilon & = & \\
+\!\!\!\!\!\!\int\limits_{0}^{\kappa} (1 \triangleleft p \circ f\_ \triangleright 0)\ \partial\varepsilon & &
\end{array}
$$

Chapter 4

Formal integration

The synthesis of homomorphisms is closely related to the well–known technique of formal differentiation (e.g. [Sha82,PK82,Pai86,Mee87b]). For each change to a source object (operator in the signature) the corresponding change in the target object is specified. In [Sha81] the analogy between incremental computation of objects and conventional integration is highlighted. He stamps the technique: Formal Integration. The integrates introduced in the preceding section exactly capture this phenomenon. The analogy to the fundamental theorem of calculus is depicted in figure 4.1.

One might argue that the correspondence is merely notational. To a large extent this is true. Integration is only a metaphor in reasoning about loops and recursion patterns. There is some resemblance in the laws, though. Compare the chain rule and the composition theorem (Th. 63):

$$G = \int g(t)dt \qquad G = \otimes\!\!\int_{c_0} g\ \partial\varepsilon$$

$$H = \int h(t)dt \qquad H = \odot\!\!\int_{c_1} h\ \partial\vartheta$$

$$H \circ G\ x = H \circ G(0) + \int_0^x (hG(t)) \times g(t)dt \qquad H \circ G\ x = \odot\!\!\int_{Hc_0} Hg\ \partial\varepsilon\ x$$

$G(x) = G(y) + \int_y^x g(t)dt$	$F\ x = \oplus\!\!\nwarrow\!\!\!\!\swarrow_{c_1} f\ \partial\beta\ x$
G	F
$+, \int$	$\oplus\!\!\nwarrow\!\!\!\!\swarrow$
y	c_0
$G(y)$	c_1
g	f
dt	$\partial\beta$

Figure 4.1: Correspondence between integrates and conventional integration.

Definition 72 (*differentiable*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$. Let $F : s \to u$.
F is *differentiable on* $\mathcal{S}_0$ if there are

(i) $\Sigma_1 : (d, u, f, \otimes, c_1)$, $\mathcal{S}_1 : \Sigma_1$

(ii) a terminable divide–function ε on $\mathcal{S}_0$,

such that $F = \otimes\!\!\nwarrow\!\!\!\!\swarrow_{c_1} f\ \partial\varepsilon$.

Not all functions are differentiable in the sense defined above. An example is the problem of the second largest in a list of integers:

$$lbo\ x := \uparrow\!/\ (\uparrow\!/\ x \neq) \lhd x.$$

However, any function tupled[1] with the identity function is differentiable. At first sight, this trivial differentiation seems a silly trick, but as shown in [Mee87b], it is sometimes a useful technique.

Proposition 73 (*trivial differentiation*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$. Let $F : s \to u$.
$(F, \mathrm{id}) : s \to s \times u$ is differentiable on $\mathcal{S}_0$.

[1] *Tupling* stands for the pairwise application of two functions to the same argument: $(F, G)\ x := (F\ x, G\ x)$.

Proof: Let $\Sigma_1 : (d, u \times s, f, \odot, c_1)$, $\mathcal{S}_1 : \Sigma_1$, where $c_1 := (Fc_0, c_0)$, $f := (F \circ \hat{}, \hat{})$ and $(r_0, x_0) \odot (r_1, x_1) := (F(x_0 \oplus x_1), x_0 \oplus x_1)$. Let σ be the broadest divide–function from proposition 49. $(F, \mathbf{id}) = \underset{c_1}{\odot\!\!\!\!\!\!\int} f \; \partial\sigma$. □

Convention 74 (*omission of the target–structure of an integrate*)
Since the target–structure is clear from the notation for integrates, it may be omitted whenever convenient.

Integrate composition is a general form of vertical loop fusion. Of course, horizontal fusion does also apply to integrates.

Proposition 75 (*integrate tupling*)
Let $\Sigma_0 : (d, s, \hat{}, \oplus, c_0)$, $\mathcal{S}_0 : \Sigma_0$. Let $F : s \to u$ and $G : s \to v$ be differentiable on $\mathcal{S}_0$, with

$$F = \underset{c_1}{\odot\!\!\!\!\!\!\int} f \; \partial\varepsilon \;, \quad G = \underset{c_2}{\otimes\!\!\!\!\!\!\int} g \; \partial\varepsilon \;.$$

Then $(F, G) : s \to u \times v$ is differentiable on $\mathcal{S}_0$.

Proof: Let $\Sigma_1 : (d, u \times v, (f, g), \oslash, (c_1, c_2))$, $\mathcal{S}_1 : \Sigma_1$, where

$$(fr_1, gr_1) \oslash (fr_2, gr_2) := (fr_1 \odot fr_2, gr_1 \otimes gr_2),$$

then $(F, G) = \underset{(c_1, c_2)}{\oslash\!\!\!\!\!\!\int} (f, g) \; \partial\varepsilon$. □

4.1 Applications

In this section, two applications of formal integration will be considered. They are formulated as propositions that might become part of a 'theory of sets'.

4.1.1 Integrate products

First, we consider problems of the form

$$H \; x = (F \; x) \otimes (G \; x)$$

where F and G are integrates. Such expressions will be called *integrate products*. A possible application might be the incremental computation of

$$H\ x = \sum_{x \in s}(f\ x) \times \sum_{x \in s}(g\ x)$$

(f and g having a number codomain).

Proposition 76 (*integrate product*)
Let $\Sigma : (d, s, \hat{}, \odot, c_0)$, $\mathcal{S} : \Sigma$. Let $F = \oplus\!\!\!\!\!\!\int_{c_1} f\ \partial\varepsilon$, $G = \oplus\!\!\!\!\!\!\int_{c_2} g\ \partial\varepsilon$, both determinate. Let $\oplus$ be associative and commutative, and let $\otimes$ be an operator that distributes through $\oplus$.

$$(F\ x) \otimes (G\ x) = \pi_1 \oslash\!\!\!\!\!\!\int_{(c_1 \otimes c_2, c_1, c_2)} ((\otimes)(f,g), f, g)\ \partial\varepsilon\ x$$

where

$$(r_0, r_1, r_2) \oslash (s_0, s_1, s_2) := (r_0 \oplus (r_1 \otimes s_2) \oplus (r_2 \otimes s_1) \oplus s_0,\ r_1 \oplus s_1,\ r_2 \oplus s_2).$$

Proof: Differentiate H on $\mathcal{S}$:

- $H\ c_0 = c_1 \otimes c_2$
- $H\ \hat{a} = (f\ a) \otimes (g\ a)$
- Let $p_\varepsilon(t_0 \odot t_1)$ hold. The parenthesis dispelling convention from [Mee86] will be used. An expression of the form "$\alpha;\beta$" stands for "$(\alpha)\beta$".

$$\begin{array}{ll} H\ t_0 \odot t_1 & = \\ F\ t_0 \odot t_1; \otimes G\ t_0 \odot t_1 & = \\ (Ft_0; \oplus Ft_1) \otimes (Gt_0; \oplus Gt_1) & = \\ (Ft_0; \otimes Gt_0) \oplus (Ft_0; \otimes Gt_1) \oplus (Ft_1; \otimes Gt_0) \oplus (Ft_1; \otimes Gt_1) & = \\ Ht_0 \oplus (Ft_0; \otimes Gt_1) \oplus (Ft_1; \otimes Gt_0) \oplus Ht_1 & \end{array}$$

Using proposition 73:

$$(H, \mathbf{id}) = \underset{(c_1 \otimes c_2, c_0)}{\ominus\!\!\!\!\!\!\int} ((\otimes)(f,g), \hat{}) \; \partial\varepsilon$$

where

$$(r_0, x_0) \ominus (r_1, x_1) := (r_0 \oplus (Fx_0; \otimes Gx_1) \oplus (Fx_1; \otimes Gx_0) \oplus r_1, x_0 \odot x_1).$$

Now one may apply integrate tupling (prop. 75) and common subexpression elimination (also known as the 'abstraction strategy') to get the desired result. □

It may seem that this result violates the differentiation metaphor, since one would expect something like

$$F(t) \cdot G(t) = \int_0^t F(t) \cdot g(t) + G(t) \cdot f(t) dt.$$

However, the analogy emerges if t_1 from the proof of proposition 76 equals $\hat{a}$, for in that case

$$H\ x \odot \hat{a} = H\ x; \oplus(Fx; \otimes ga) \oplus (fa; \otimes Gx) \oplus (fa; \otimes ga).$$

The last product vanishes in conventional integration because of the infinitesimal steps.

One final remark on this example is in order. The integrate product from the proposition does not excel in typographical clarity. This problem is common to most expressions that result from tupling. Yet the technique is important [Pet84] and the results merit a more transparent notation. It is not obvious, however, what it should look like.

4.1.2 Set partitions

A second application of the approach discussed in this paper are partition problems on sets. A partition of a set x is a collection c of pairwise disjoint subsets of x such that $(\cup/c = x)$. The set of all partitions of x is denoted by $(setparts\ x)$. In example 66, we encountered a function that yields some partition of x. Starting from that definition, the following result can be calculated.

Proposition 77 (*set partitions*)
Let $\theta\ x = \oplus/f * setparts$, where $f = \otimes/g*$. If $\otimes$ distributes through $\oplus$, then

$$\begin{aligned} \theta\ \emptyset &= g\emptyset \\ \theta\ x &= \oplus/((\otimes)\theta^2) * divs^+ x;\ \oplus(g\ x; \otimes g\ \emptyset) \oplus (g\ x) \end{aligned}$$

where

$$\begin{aligned} \theta^2(x,y) &= (\theta x, \theta y) \\ divs^+ x &= \{(u,v) | u \cup v = x, u \cap v = \emptyset, u \neq \emptyset, v \neq \emptyset\} \end{aligned}$$

In his theory of lists, Bird attained useful results on list partitions [Bir86b]. The above proposition might be the basis for a similar theory on sets [dM88]. Due to time and space limits, its proof cannot be elaborated here.

Chapter 5

Discussion

The research described in this paper is a modest attempt to clarify the role of 'divide' in the development of divide–and–rule algorithms. The formal background that was developed to this end might be of use in the understanding of morphic programming as a whole.

A promising area to put integrates to use is a future theory of sets. Few operations beat set union in algebraic richness, so many operations will need a divide function. Also, it is often convenient to take symmetric set difference as the constructing operator, rather than union [Sha82]. Our treatment of structures facilitates such a change of view.

Morphic programming is by no means restricted to the binary structure types considered here. However, until the present day little work has been done on general structure preserving maps [Mee87c]. The approach taken in this report is not heavily dependent on the binary nature of structures, and could probably be generalized to other structure types. The proof of the composition theorem suggests that similar results can be automatically derived.

There are a few unresolved problems concerning the work presented here. The mathematical framework as introduced in section 3.1 is still shaky. A more rigorous treatment, especially of indeterminate choice and refinement, should be pursued. If indeterminacy is excluded, it should be possible to give an elegant treatment of integrates using Zantema's approach [Zan87]. His theory would need a generalization to partial operations.

Another problem is of a typographical nature. The notation used for

integrates is — to say the least — baroque. Considering future generalization, it might be wiser to employ a notation like

$$(\hat{}, \oplus, c_0)_e := (\tilde{}, \otimes, c_1).$$

However, this is unwieldy too, and it does not fit into the conventional squiggles employed in morphic programming.

The best way to proceed from here is putting both notation and theory to practice. Some basic results on set problems seem to be within reach, so that is where efforts should focus. If these investigations affirm the practical merits of integrates, a next step should be the generalization to other structure types.

Acknowledgements

I would like to express my thanks to S.G. van der Meulen for drawing my attention to the subject of algorithmics. Doaitse Swierstra taught me the basics of the subject and encouraged me to write this study. His support and advice have been invaluable. Many helpful remarks were provided by Lambert Meertens, Jeroen Fokker and Nico Verwer. Finally, I want to thank Hans Zantema for his thorough criticism and numerous suggestions.

Bibliography

[BBB*85] F.L. Bauer, R. Berghammer, M. Broy, W. Dosch, F. Geiselbrechtinger, R. Gnatz, E. Hangel, W. Hesse, B. Krieg–Brückner, A. Laut, T. Matzner, B. Möller, F. Nickl, H. Partsch, P. Pepper, K. Samelson, and M. Wirsing. *The Munich project CIP, vol. I: The Wide Spectrum Language CIP–L.* Volume 183 of *Lecture Notes in Computer Science*, Springer–Verlag, 1985.

[Bir86a] R.S. Bird. Exercises on the theory of lists. NFI seminar Utrecht University, December 1986.

[Bir86b] R.S. Bird. *An Introduction to the Theory of Lists.* Technical Monograph PRG–56, Oxford University Computing Laboratory, Programming Research Group, October 1986.

[BK80] M. Broy and B. Krieg–Brückner. Derivation of invariant assertions during program development by transformation. *ACM Transactions on Programming Languages and Systems*, 2(3), 1980.

[BMW85] R.S. Bird, L.G.L.T. Meertens, and D.S. Wile. *A Common Basis for Algorithmic Specification and Development.* Working paper ARK–3, IFIP WG 2.1, 1985.

[BPW80] M. Broy, P. Pepper, and M. Wirsing. On relations between programs. In B. Robinet, editor, *International Symposium on Programming*, pages 59–78, Springer–Verlag, 1980.

[dM88] O. de Moor. Set partitions. In preparation, scheduled to appear, 1988.

[Mee86] L.G.L.T. Meertens. Algorithmics — towards programming as a mathematical activity. In J.W. de Bakker, M. Hazewinkel, and J.K. Lenstra, editors, *Mathematics and Computer Science*, pages 289–334, CWI Symposium, North–Holland, 1986.

[Mee87a] L.G.L.T. Meertens. *An Abstracto reader prepared for IFIP WG 2.1.* Note CS–N8702, Centre for Mathematics and Computer Science Amsterdam, Department of Algorithmics & Architecture, April 1987.

[Mee87b] L.G.L.T. Meertens. *Formal Differentiation — A Page from a Book on Algorithmics.* Working paper 550 COR–5, IFIP WG 2.1, 1987.

[Mee87c] L.G.L.T. Meertens. Towards general morphisms. Note Utrecht Algorithmics Club, 1987.

[MG83] J. Meseguer and J.A. Goguen. Initiality, induction and computability. In M. Nivat and J.C. Reynolds, editors, *Algebraic methods in semantics*, pages 460–541, Cambridge University Press, 1983.

[Mol85] B. Möller. On the algebraic specification of infinite objects — ordered and continuous models of algebraic types. *Acta Informatica*, 22:537–578, 1985.

[Mor87] C. Morgan. Refining non–deterministic expressions. July 1987. Draft.

[Pai86] R. Paige. Programming with invariants. *IEEE Software*, 56–69, January 1986.

[Pet84] A. Pettorossi. *Methodologies for program transformation and memoing.* PhD thesis, Computer Science Department, Edinburgh University, 1984.

[PK82] R. Paige and S. Koenig. Finite differencing of computable expressions. *ACM Transactions on Programming Languages and Systems*, 4(3):402–454, 1982.

[Sha81] M. Sharir. Formal integration: a program transformation techniqe. *Computer Languages*, 6:35–46, 1981.

[Sha82] M. Sharir. Some observations concerning formal differentiation of set theoretic expressions. *ACM Transactions on Programming Languages and Systems*, 4(2):196–225, 1982.

[Smi85] D.R. Smith. The design of divide and conquer algorithms. *Science of Computer Programming*, 5:37–58, 1985.

[Smi87a] D.R. Smith. Applications of a strategy for designing divide–and–conquer algorithms. *Science of Computer Programming*, 8:213–229, 1987.

[Smi87b] D.R. Smith. On the design of generate–and–test algorithms: subspace generators. In L.G.L.T. Meertens, editor, *Program Specification and Transformation*, pages 207–220, IFIP, Elsevier Science Publishers B.V., 1987.

[vD83] D. van Dalen. Logic and structure. 1983.

[Wad87] P. Wadler. Views: a way for pattern matching to cohabit with data abstraction. In *Proc. Principles of Programming Languages 14*, pages 307–313, ACM, 1987.

[WPP*83] M. Wirsing, P. Pepper, H. Partsch, W. Dosch, and M. Broy. On hierarchies of abstract data types. *Acta Informatica*, 20:1–33, 1983.

[Zan87] H. Zantema. Towards algorithmics as a mathematical activity. Note Utrecht Algorithmics Club, 1987.

Index

abstract type 5
algebraic richness 19
    congruence class 32
    restriction 19
apply–to–all 28
breadth 8
broadest divide function 38
carrier set 7
composition 49
homomorphism 57
    integrate and homomorphism 53
congruence class 32
    granularity 32
constant 15
conversion 46
    conservative 47
convertible 46
    no junk 47
determinate 10
    indeterminate inserted–in 35
    integrate 41
    specialization 43
differentiation 61
    trivial 61
divide function 36
    broadest 38
    terminable 37
domain 15
    initial algebra 20
embedding 15
equivalent 9
formal differentiation 61
formal integration 61
functionality 5
ground expressions 5
homomorphism 22
    condition 45
    composition 57
    initiality 24
    integrate 43
    poorer 24
    promotion 57
indeterminate inserted–in 33
    determinate 35
induction 18
initial algebra over a domain 20
    homomorphism 24
initial model 7
    algebra over a domain 20
initiality 20
    homomorphism 24
inserted–in 26
    indeterminate 33, 35
integrate 39
    composition 49, 53
    determinate 41
    homomorphism 43

inversion 52
product 63
tupling 62
integration 61
inversion 52
language 5
laws 5
main carrier 15
manageable structure 20
model 7
monotonic 11
operation name 5
operation 7
operator 15
parsed congruence class 37
parsing predicate 36
partition 65
poorer 23
homomorphism 24
reflexivity 23
transitivity 23
product 63
projection 24
reduct 7
refinement 10
restriction 6
algebraic richness 19
satisfy 9
set partition 65
signature 5
structure 12
surface 18
sorted 5
sorts 5
specialization 42
determinate 43
structure 14
manageable 20
signature 12
structure signature 12
surface signature 18
term translation 22
terminable divide function 37
theory 6
total 10
tupling 62
valuation 7
visible subset 17
structure signature 17
visible terms 15