A Mathematical Approach to Nondeterminism in Data Types

WIM H. HESSELINK
University of Groningen

The theory of abstract data types is generalized to the case of nondeterministic operations (set-valued functions). Since the nondeterminism of operations may be coupled, signatures are extended so that operations can have results in Cartesian products. Input/output behavior is used to characterize implementation of one model by another. It is described by means of accumulated arrows, which form a generalization of the term algebra. Morphisms of nondeterministic models are introduced. Both innovations prove to be powerful tools in the analysis of input/output behavior. Extraction equivalence and observable equivalence of values are investigated. Quotient models for such equivalence relations are constructed. The equivalence relations are compared with each other, with separation of values by means of experiments, and with the separation property that characterizes a terminal model. Examples are given to show that the four concepts are different. In deterministic models the concepts coincide.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*semantics*; D.3.3 [**Programming Languages**]: Language Constructs—*abstract data types*; *data types and structures*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*automata*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*alternation and nondeterminism*; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*type structure*

General Terms: Design, Languages, Theory

Additional Key Words and Phrases: Abstract data type, behavioral equivalence, distinguishable, extraction equivalence, nondeterminism, nondeterministic data type, observable equivalence, signature, term algebra, terminal model, value consistency

1. INTRODUCTION

1.1 Nondeterministic Data Types

This paper develops a theory of data structures in which correctness of an implementation can be formally verified, without imposing irrelevant restrictions to implementations. Proving correctness is a formal activity. Therefore we must be able to lift an arbitrary implementation to the formal level. In order not to be forced to specify all details of an implementation, we have to admit nondeterminism on the formal level (cf. [2, Sect. 4.4]). It is especially useful in treating

Author's address: University of Groningen, Department of Mathematics and Computing Science, P.O. Box 800, 9700AV Groningen, The Netherlands.

things like memory allocation, overflow conditions, and uninitialized fields of structured values.

The leading principle is to avoid premature or unnecessary design decisions. Therefore we have a preference for loose specifications, which admit a variety of nonisomorphic models (cf. [17, sect. 0]). We shall define a *nondeterministic data type (NDDT) to be a variety of nonisomorphic nondeterministic models of one* signature. In this definition the specification is deliberately omitted, because we want to separate the specification from the object to be specified.

This paper is an investigation of NDDTs, and not of specification methods. We aim at understanding arbitrary NDDTs, so that in the design of useful NDDTs one can avoid "the slings and arrows of outrageous nondeterminism." We consider nondeterminism as the rule, and determinism as a property that may be specified (cf. [3]). References [10] and [16], our main sources on NDDTs, are written from the point of view that the specification determines the set theoretic structure of the models (cf. [10, p. 23]). As a result, nondeterminism is treated there as an admissible exception rather than as a rule (cf. [2, p. 328]).

This paper is a complete revision of an earlier paper [7], which was written in ignorance of all related work on NDDTs, multialgebras, and observable equivalence. The most drastic changes, however, are due to the referee who pointed out that a correctness criterion for implementations was badly needed.

1.2 Implementation

The main theme of this paper is the concept of implementation. In order to prevent misunderstanding, we must first distinguish two aspects of the general concept of implementation:

(1) *The vertical aspect.* In this case two signatures $\sum_0$ and $\sum_1$ are given. The implementation consists of a construction of a model A_1 of $\sum_1$ by means of a model A_0 of $\sum_0$. The values of A_1 are structured values of A_0. The operations of A_1 are routines that use the operations of A_0. Hidden values of A_0 can give rise to nondeterminism in A_1.

(2) *The horizontal aspect.* This aspect is based on the idea of a user of a black box, who specifies external values, applies operations, rearranges unknown internal values, and observes resulting external values. The box represents a given signature $\sum$. It is supposed to contain some model of $\sum$. A model A of $\sum$ is considered as an implementation of a model B of $\sum$, if all finite sequences of experiments with model A in the box give results that could have been produced by model B.

Henceforth in this paper, we use the term *implementation* exclusively in the horizontal sense. So it is defined in terms of experiments. In order to get a more applicable definition, we shall make an analysis of input/output behavior. With the resulting correctness criterion for implementations, it is easy to falsify but difficult to verify whether a model A is an implementation of a model B. For the latter purpose, we introduce morphisms of nondeterministic models. The existence of a morphism M: $A \longrightarrow B$ is a sufficient condition for A to implement B, but not a necessary condition.

1.3 Separation of Values

An important aspect of implementations of abstract data types is that different values of the implementation can represent the same abstract value. In our presentation we do not have such abstract values, but nevertheless different values of a model may have the same observable behavior. In such a case, it can be useful to treat behaviorally equivalent values as if they were identical. This requires the construction of a quotient model. The second part of this paper is devoted to the theory of equivalence relations and quotients in the nondeterministic case. It turns out that there are two adequate equivalence relations, and two related separation criteria. It is proved that the quotient models have the expected properties.

1.4 Contents of the Paper

The basic definitions concerning nondeterminism, signatures, models, and data types are collected in Section 2. As the nondeterminism of different operations may be coupled, signatures are extended so that operations can have results in Cartesian products, and given external domains are incorporated in the signature. In Section 3 we present an example that is a simple case of storage management. The axioms of this example are shown to be complete with respect to certain intentions of a user.

In Section 4 we introduce accumulated arrows (a generalization of the concept of term algebra), and implementation and equivalence of models (cf. (2) of Section 1.2). Section 5 is devoted to morphisms of nondeterministic models. These are more adequate than the homomorphisms of [10]. Some simple but crucial examples of implementations are given. In Section 6 it turns out that our morphisms lead to a useful concept of terminal models. If a data type has a terminal model, then this model may serve as a specification for all models of the type. As an example we treat the data type bag, a common generalization of stack and queue.

Section 7 is devoted to the theory of extraction equivalence of values (cf. [16]). Equivalence relations are used to identify equivalent values. This leads to the concept of quotient model. In the case of extraction equivalence, the construction of the quotient model is new, and so is the theorem in Section 7.4, that a model is implementation equivalent to its quotient under extraction equivalence. The proof of this result is surprisingly delicate.

The slightly stronger concept of observable equivalence (cf. [10]), is treated in Section 8. In [10] Kapur gave a recursive definition of observable equivalence, or rather of its negation. By a variation of Tarski's fixed point theorem, we prove that the recursive equation indeed has a solution.

Section 9 contains a discussion of inseparability and consistency of values. Two values are inseparable if there is no experiment that always shows a difference between the two. Consistency is a little bit stronger. It is proved in Theorem 9.1 that a model is terminal if and only if consistent values of the model are always equal.

In Section 10 inseparability and consistency are compared with the two equivalence relations. Examples are provided to show that the set of implications derived is complete. Some readers may prefer to look at these examples earlier, but a full discussion of the examples requires all the preceding sections.

1.5 Summary of the Main Results

Here we list the main innovations of this work with respect to the papers by Kapur [10] and Subrahmanyam [16]:

(1) The accumulated arrow (Section 4.2) is a new and powerful formalism that enables us to deal with operations that modify more than one argument.

(2) We give a new formalization of the concept of implementation (Section 4.3).

(3) Morphisms are an important new tool, especially in the study of implementations (see Section 5).

(4) In Sections 7–10 we provide a mathematical theory of behavior in NDDTs, in which the statements of [10] and [16] have been made rigorous and have been extended. Comparison of their concepts leads to yet two other separation concepts. This sheds new light upon the notion of behavior for nondeterministic mechanisms.

1.6 Notations for Sets, Functions, and Case Distinctions

A finite set X with the elements $x_1, \ldots, x_n$ is denoted by $X = \{x_1, \ldots, x_n\}$. The empty set is $\varnothing$. We say that x is in X, or $x \in X$, to indicate that x is an element of the set X. We say that Y is contained in X to mean that Y is a subset of X. The subset of X consisting of the elements x that satisfy a condition $P(x)$ is denoted by $\{x \in X \mid P(x)\}$. We use the standard sets

$$\begin{aligned} \text{BOOL} &= \{\textbf{true}, \textbf{false}\}, \\ \text{NULL} &= \{0\}, \\ \text{NAT} &= \{0, 1, 2, \ldots\}, \\ \text{INT} &= \{\ldots, -2, -1, 0, 1, 2, \ldots\}. \end{aligned}$$

We use lambda abstraction to denote mappings

$$f = \lambda x.f(x).$$

In a formalized case analysis, we use Dijkstra's guarded expressions

$$\textbf{if} \rightarrow [\!] \rightarrow \textbf{fi}.$$

2. DEFINITIONS FOR NONDETERMINISTIC DATA TYPES

In this section we present our basic definitions. Since they are not completely standard, we start with an informal discussion. We fix our concepts for maps and nondeterministic operations. After laying down some conventions, we give the formal definitions, followed by a comparison with definitions in related work.

2.1 Introduction to Models, Signatures, and Data Types

Let A be an implementation of a data structure, say, of a stack of integers. It associates with each of the sorts **stack** and **integer** a set of values: $\textbf{stack}^A$, the set of the possible states of the stack; and $\textbf{integer}^A$, the set of the implemented integers. It also associates with each of the symbols **newstack**, **push**, and **pop** an operation

$$\begin{array}{lccl} \mathbf{newstack}^A : & \text{NULL} & \to & \mathbf{stack}^A, \\ \mathbf{push}^A : & \mathbf{integer}^A \times \mathbf{stack}^A & \to & \mathbf{stack}^A, \\ \mathbf{pop}^A : & \mathbf{stack}^A & \to & \mathbf{integer}^A \times \mathbf{stack}^A. \end{array}$$

The mathematical prescription A that associates with a symbol x or f a set x^A or an operation f^A is called a model. We allow Cartesian products as domains and as codomains of the operations. The functionalities of an operation f are described by strings of sorts, the domain and codomain strings dom(f) and cod(f).

It is useful to be able to express the fact that certain domains are given. For example, in the stack of integers the set of the implemented integers is usually given. In that case we say that **integer** is an external sort with a fixed external domain, say, INT, and all models A are supposed to satisfy $\mathbf{integer}^A = \text{INT}$. The formal structure consisting of the sorts, the arrow symbols, the domain and codomain designators, and the external sorts and domains is collected in the concept of signature.

Usually, one does not want to consider all models of a given signature, but only those that satisfy certain rules. At this stage, however, we do not want to specify the language of the permissible rules. We accept every class of models of a signature as a genuine data type.

2.2 Maps and Operations (Nondeterminism)

We use the terms *map*, *mapping*, and *function* synonymously. We introduce a formal symbol **error**, which is not an element of any of the sets we start with. By a partial map $f: X \to Y$, we mean an ordinary map $f: X \to Y^+$, where Y^+ is the disjoint union $Y \cup \{\mathbf{error}\}$. (Since Y is not considered as a **cpo** (cf. [15]), we do not use the term *bottom*. The symbol **undefined**, however, could be used just as well.)

Informally speaking, an operation $f: X \to Y$ is a device that attaches nondeterministically to an element $x \in X$ some resulting element $y \in Y$ or the result **error**. So f is specified by associating with each element $x \in X$ the set of the possible results. This set is denoted by $f(x)$. It is a nonempty subset of Y^+.

Definition. An *operation* $f: X \to Y$ is defined to be a map $f: X \to Q(Y)$, where $Q(Y)$ is the set of the nonempty subsets of Y^+. For convenience, we define $f(\mathbf{error}) = \{\mathbf{error}\}$ for every operation f. So **error** is propagated (cf. [5]), and the operation can be viewed as a map $X^+ \to Q(Y)$. If A is a subset of X^+, we use $f(A)$ to denote the union of the sets $f(x)$ with $x \in A$. If A is nonempty, so is $f(A)$.

An operation $f: X \to Y$ is said to be *deterministic* if for every element $x \in X$ the set $f(x)$ contains precisely one element, possibly the element **error**. An operation $f: X \to Y$ is said to be *total* if **error** is not in $f(x)$ for any element $x \in X$.

Every partial map $g: X \to Y$ can be identified with a unique deterministic operation $f: X \to Y$ by putting $f(x) = \{g(x)\}$. Conversely, a deterministic operation determines a partial map. The ordinary mappings $f: X \to Y$ correspond bijectively to the operations that are both deterministic and total.

Remark. It is an important decision to require that the result sets $f(x)$ are nonempty for every operation f. This condition is equivalent to the law of the excluded miracle (cf. [3]). It is the only liveness condition we impose.

2.3 Notations for Strings and Cartesian Products

If S is a set of symbols (sorts), then S^* denotes the set of all finite strings of elements of S. The empty string is denoted by ε. Concatenation of strings is denoted by means of the infix operator $+$. If A is a prescription that assigns to every element $s \in S$ a set s^A, and if $p = s_1 \cdots s_n$ is a string in S^*, then we write p^A to denote the Cartesian product set

$$p^A = s_1^A \times \cdots \times s_n^A.$$

So, the set p^A consists of the elements $x = (x_1, \ldots, x_n)$ with $x_i \in s_i^A$. The Cartesian product of an empty string of sets is (by convention) the one-point set NULL = $\{0\}$. So, we always have $\varepsilon^A =$ NULL.

2.4 Signature, Model, Value, and Type

Definition. A *signature* $\sum = (S, F, \text{dom}, \text{cod}, E, K)$ consists of a set S of sorts, a set F of arrow symbols, two mappings dom, cod: $F \longrightarrow S^*$, a subset E of S, and a prescription K that assigns to every element $s \in E$ a set s^K. The elements of E are called the *external* sorts, and the sets s^K are called the *external* domains. We say that $f: p \longrightarrow q$ is an arrow of $\sum$ to indicate that f is in F and that $\text{dom}(f) = p$ and $\text{cod}(f) = q$ in S^*.

In examples, a signature is given as follows: In the denotation of the set S, every external sort is followed by the symbol := and the external domain s^K. So $e :=$ ABC means $e \in E$ and $e^K =$ ABC. In the denotation of F, we write the arrows and not only the arrow symbols. In this way the data E, K, dom, and cod are incorporated in the denotations of S and F. For an example, see Section 3.1.

Definition. A *model* A of $\sum$ is specified by associating with every sort $s \in S$ a set s^A such that $s^A = s^K$ whenever s is in E, and associating with every arrow $f: p \longrightarrow q$ of $\sum$ an operation $f^A: p^A \longrightarrow q^A$. Note that p and q are strings, so that p^A and q^A are Cartesian products (cf. Section 2.3). A *value* of a model A is defined to be a pair (x, p) with $p \in S^*$ and $x \in p^A$. Usually, the value (x, p) is identified with the element x. One has to be careful, however, if the sets s^A are not disjoint.

A string of sorts p is said to be *external* if p consists of external sorts. It is called *internal* otherwise. A value $x \in p^A$ is said to be *external* (*internal*) if p is external (internal).

A *nondeterministic data type* (*NDDT*) is defined to be a pair $(\sum, T)$, where $\sum$ is a signature and T is a class of models of $\sum$.

2.5 Comparison with Definitions in Related Work

Data types used in practice frequently have procedures with more than one output parameter. In almost all theoretical work on abstract data types, only one output parameter is permitted. One of our results is that this restriction is not essential. Formally speaking, let a signature $\sum$ be called *focused* if the codomain string $\text{cod}(f)$ of every arrow f has length one. In [10, p. 26], it is argued that a reduction to a focused signature is always possible, either by modeling an operation with results in a Cartesian product as a number of separate operations or by introducing the Cartesian product as a new type. In the nondeterministic case, the first alternative is not satisfactory, as it does not allow coupling between the nondeterministic choices (cf. [8, 1.3.3]). The second alternative is theoretically sound, but has the disadvantage of introducing new sorts, new arrows, and new axioms. Therefore we have chosen to work with not necessarily focused signatures. For convenience, we have incorporated external sorts and domains in the signature. These correspond to the visible sorts of [4] and the global sorts of [16].

Let a model A of $\sum$ be called *deterministic* (*total*) if all its operations are deterministic (total). The $\sum$-algebras of [17] are deterministic models of focused signatures. The data types of [16] are total models of focused signatures. In fact, [10] and [16] are our only sources with nondeterministic models. The concept of multialgebra (cf. [13]) corresponds to a nondeterministic, total model of a focused signature with only one sort. As a step toward the admittance of **error**, [6] admits operations f such that the set of results $f(x)$ may be empty. However, this does not allow a nondeterministic choice between **error** (for example, overflow) and a meaningful value (see also [8, pp. 1–12]).

As for our definition of types, we refer to the introduction of [17] for arguments leading to the admittance of a wide class of axioms and a variety of nonisomorphic models. In fact, we go further: We put no conditions on the class T of an NDDT. Consequently, we have all the freedom to introduce hidden or auxiliary functions in the axioms that determine a data type. For an example see Section 6.4.

3. AN EXAMPLE OF NONDETERMINISTIC STORAGE MANAGEMENT

In this section we show that our formalism is sufficiently rich for an elegant formalization of a simple case of storage management. We specify an NDDT pointer table, which models a table of pointers to items, such that every item can get a unique position in the table. One may think of a lexicographic tree or a hash table (cf. the programs 4.5 and 4.8 of [18]). In most applications the position in the table is also used to attach certain attributes to the items. This extension, however, gives no extra complications.

The example shows that our extension of the usual definitions is not empty. In fact, the signature is not focused, and many relevant models are not total and not deterministic. In [8, 1.3.2] a similar example is given, attributed to L. Morris. The class T of the acceptable models is characterized by axioms. We show that the axioms are complete with respect to certain formalized intentions of the user.

3.1 The Data-Type Pointer Table

Let ITEM be a given set of values (the items). The signature $\sum$ is given by

$$\begin{array}{lllll} S = \{ & \mathbf{item} := \mathrm{ITEM},\ \mathbf{table},\ \mathbf{pointer}\ \}, & & & \\ F = \{ & \mathbf{create} & : & \varepsilon & \rightarrow \mathbf{table}, \\ & \mathbf{key} & : & \mathbf{pointer\ table} & \rightarrow \mathbf{item}, \\ & \mathbf{position} & : & \mathbf{item\ table} & \rightarrow \mathbf{pointer\ table}\ \}. \end{array}$$

The data-type pointer table is the pair ($\sum$, T), where T is the class of the models A of $\sum$ that satisfy the following axioms:

(1) The operation $\mathbf{key}^A$: $\mathbf{pointer}^A \times \mathbf{table}^A \rightarrow$ ITEM is deterministic (so it may be considered as a partial map (cf. Section 2.2)).

(2) Let $t \in \mathbf{create}^A(0)$. Then $t \neq \mathbf{error}$, and $\mathrm{key}^A(p, t) = \mathbf{error}$ for every element $p \in \mathbf{pointer}^A$.

(3) Let $\mathbf{key}^A(p, t) = \mathbf{key}^A(q, t)$, where $p, q \in \mathbf{pointer}^A$ and $t \in \mathbf{table}^A$. Then $p = q$ or $\mathbf{key}^A(p, t) = \mathbf{error}$.

(4) Let (p, t_1) be in $\mathbf{position}^A(x, t_0)$. Then we have

   (a) $\mathbf{key}^A(p, t_1) = x$; and

   (b) $\mathbf{key}^A(q, t_1) = \mathbf{key}^A(q, t_0)$ whenever $q \neq p$ or $\mathbf{key}^A(q, t_0) \neq \mathbf{error}$.

Remark. The operation $\mathbf{position}^A$ is always permitted to yield **error**, which means overflow. On the present level of abstraction, we do not want to specify acceptable overflow conditions. Note that the assumption in axiom 4 implies that no **error** has been delivered.

3.2 Intuitive Completeness

It is not obvious that all our implicit assumptions about tables of pointers to items are implied by the above axioms. As long as the assumptions are implicit, such a thing cannot be verified. Therefore we formalize the intended interpretation of the elements t of the sets $\mathbf{table}^A$. The interpretation of t is defined to be the operation

$$h\colon \mathbf{pointer}^A \rightarrow \text{ITEM} \qquad \text{with} \qquad h(p) = \mathbf{key}^A(p, t).$$

Axiom 1 requires that h is a partial function. Axiom 2 says that the operation $\mathbf{create}^A$ always yields a table t that is interpreted as the function h with an empty domain of definition. Axiom 3 says that the interpretation h of any table t must be injective on its domain of definition (so that it is a bijection between the occurring pointers and the occurring items). As for axiom 4, let (p, t_1) be in $\mathbf{position}^A(x, t_0)$, and let h_0 and h_1 be the interpretations of t_0 and t_1, respectively. Let

$$V_i = \{q \in \mathbf{pointer}^A \mid h_i(q) \neq \mathbf{error}\}$$

be the domain of definition of h_i. Axiom 4 says that V_1 is the union of V_0 and $\{p\}$, that $h_1(p) = x$, and that h_1 and h_0 agree on the set V_0.

This shows that the interpretation h of every table t that can be constructed is determined inductively, apart from the choices of the new pointer values. So the axioms are complete with respect to the intended interpretations of the tables.

3.3 Special Mathematical Models

The analysis of Section 3.2 suggests certain special models in which the tables t coincide with their interpretations h as defined in Section 3.2. We construct these models as follows: Let P be an arbitrary set, let H be the set of the partial mappings $h: P \longrightarrow \mathrm{ITEM}$ that are injective on their domain of definition

$$V = \{p \in P \mid h(p) \neq \textbf{error}\},$$

and let $h_0 \in H$ be the partial map with the empty domain of definition. The model C of $\sum$ is defined by

$$\begin{aligned}
\textbf{pointer}^A &= P,\\
\textbf{table}^A &= H,\\
\textbf{create}^A(0) &= \{h_0\},\\
\textbf{key}^A &= \lambda(p, h) \cdot h(p),\\
\textbf{position}^A &= \lambda(x, h) \cdot \{\textbf{error}\}\\
&\quad \cup \{(p, k) \in P \times H \mid k(p) = x\\
&\qquad \wedge ((q \neq p \vee h(q) \neq \textbf{error}) \Rightarrow k(q) = h(q))\}.
\end{aligned}$$

It is easy to see that C belongs to class T. The operation **create**^{A} is deterministic. The operation **position**^{A} is as nondeterministic as possible. If **pointer** was an external sort with external domain P, then model C was terminal (cf. Section 6.3). Since sort **pointer** is internal, however, class T does not have a terminal model. A proof of this fact falls outside the scope of this paper.

4. ACCUMULATED ARROWS, IMPLEMENTATIONS, AND EQUIVALENCE OF MODELS

In an earlier version of this paper [7], it was suggested that an implementation of one model by another model was the same as a homomorphic relationship. This suggestion is not true, as emphasized strongly by one of the referees. Here we give a formal definition of implementation in terms of input/output behavior.

4.1 Input/Output Behavior of Models

Let A be a model of a given signature $\sum$. A user of model A has the disposal of a black box that contains the model. He commands the model by means of the signature. He only knows the names of the operations and their functionalities, and the external values. He uses the model by specifying external values, applying operations, rearranging unknown internal values, and observing external values that are delivered and errors that occur.

In order to describe this input/output behavior we need a generalization of the concept of term algebra (cf. [17]), or the derived signature (cf. [4, p. 269]). The usual term algebra is not satisfactory for two reasons: First, our signature is not necessarily focused (cf. Section 2.5). This fact adds a considerable complexity. The second reason is the nondeterminism: A copy of a result of an operation may differ from another result of the same operation with the same arguments. Therefore we start from scratch. The set of arrows F of the signature $\sum$ is extended to a set of accumulated arrows. To every model A and every accumulated arrow $f: p \longrightarrow q$, we associate an accumulated operation $f^A: p^A \longrightarrow q^A$. The input/output behavior of A is determined by the effects of the accumulated operations on the external values.

Note the following difference of our approach with other approaches. Even if one starts with a focused signature (cf. Section 2.5), the codomain string q of an accumulated arrow f: $p \longrightarrow q$ can be arbitrarily long. Therefore accumulated operations can deliver results of arbitrary complexity.

4.2 Accumulated Arrows and Accumulated Operations

Definition. The accumulated arrows of the signature $\sum$, and the accumulated operations of the model A of $\sum$ are defined inductively in three steps:

Step 1. Every arrow $f: p \rightarrow q$ of $\sum$ is an accumulated arrow. The corresponding accumulated operation $f^A: p^A \rightarrow q^A$ is the given operation f^A of model A.

Step 2. *Partial composition.* Let $f: p \rightarrow q + r$ and $g: r + s \rightarrow t$ be accumulated arrows, where + stands for concatenation (cf. Section 2.3). Then we have an accumulated arrow

$$(g[r]f): p + s \rightarrow q + t.$$

The corresponding accumulated operation

$$(g[r]f)^A: p^A \times s^A \rightarrow q^A \times t^A$$

is given as follows: A pair $(u, v) \in q^A \times t^A$ is an element of the set $(g[r]f)^A(x, y)$ if and only if there exists $z \in r^A$ such that (u, z) is in $f^A(x)$ and v is in $g^A(z, y)$. The set $(g[r]f)^A(x, y)$ contains the element **error**, if and only if $f^A(x)$ contains **error**, or $f^A(x)$ contains a pair (w, z) such that $g^A(z, y)$ contains **error**.

Step 3. *Rearrangement.* Let p and q be strings of sorts, say, $p = p_1 \cdots p_m$ and $q = q_1 \cdots q_n$. Let a mapping b: $\{1 \cdots n\} \rightarrow \{1 \cdots m\}$ be given such that $p_{b(i)} = q_i$ for all indices $i \in \{1 \cdots n\}$. Then we have an accumulated arrow

$$(q]b[p): p \rightarrow q.$$

The corresponding accumulated operation

$$(q]b[p)^A: p^A \rightarrow q^A$$

is the mapping (cf. Section 2.2) that is defined by

$$(q]b[p)^A(x_1 \cdots x_m) = (y_1 \cdots y_n) \qquad \text{with} \quad y_i = x_{b(i)}.$$

Remarks

(1) If in Step 2 the strings q and s are empty, then $(g[r]f)$ stands for the ordinary composition. If $r = \varepsilon$ (cf. Section 2.3), then $(g[\varepsilon]f)$ stands for the natural operation between the Cartesian products.

(2) Rearrangement unifies three kinds of transactions, namely, permutation of variables, copying of variables that are needed more than once, and forgetting of variables that are no longer needed. If the order of the variables does not allow a partial composition, one can first apply a rearrangement.

4.3 The Concept of Implementation

Let A and B be models of $\sum$. If $f: p \rightarrow q$ is an accumulated arrow between external strings p and q, the sets p^A and p^B are both equal to the set p^K, which is a product of external domains. Similarly, q^A and q^B are equal to q^K. Therefore we may compare the operations

$$f^A, f^B: p^K \rightarrow q^K.$$

Definition. Model A is said to be an *implementation* of B if $f^A(x)$ is a subset of $f^B(x)$ for every accumulated arrow $f\colon p \longrightarrow q$ between external strings and every value $x \in p^K$.

This definition differs completely from the definitions in [10] or [16], so it deserves a justification. Assume that a user has a black box, which contains a model of $\sum$ that is specified as the model B. Assume that the implementer of the box has provided a model A. The user has no grounds to complain as long as every accumulated operation with an arrow $f\colon p \longrightarrow q$ between external domains p^K and q^K, applied to a value $x \in p^K$, yields some value $y \in f^B(x)$. The box can deliver every element $y \in f^A(x)$. So it is guaranteed that the user cannot complain if and only if $f^A(x)$ is a subset of $f^B(x)$ in every such case.

Here, as one of the referees remarked, we assume that every possible value is acceptable under all circumstances. In particular, we do not want to guarantee that every possible value will eventually occur. So fairness requirements are excluded. In fact, we only consider finite calculations. Compare (2) of Section 1.2 and Section 4.5.

4.4 Implementation Equivalence

Definition. Two models A and B of $\sum$ are said to be *implementation equivalent* if either model is an implementation of the other model (cf. Section 4.3) Equivalently, A is implementation equivalent to B if and only if $f^A = f^B$ for every accumulated arrow $f\colon p \longrightarrow q$ between external strings.

Remark. This concept of equivalence can be characterized as behavioral equivalence with respect to inputs and outputs of observable sorts (cf. [14, 2.4]). It may happen, however, that implementation-equivalent models have different observable behavior (cf. remark (2) of Section 5.4).

4.5 An Example of Unbounded Iteration

Accumulated arrows between external strings may be considered as finite programs without control structures. One could argue that control structures should be admitted. Admission of conditional expressions or bounded iterations is harmless; as in Section 4.3 and 4.4, we consider arbitrarily complex accumulated arrows. Admission of unbounded iteration in the formalism of Section 4.2 would give a different theory. This is shown in the following example: Let $\sum$ be the signature of the shaking urn of Booleans, with

$$\begin{aligned} S = \{\ &\mathbf{urn}, \mathbf{bool} := \text{BOOL}\ \},\\ F = \{\ &\mathbf{fill}\ \ :\ \varepsilon \longrightarrow \mathbf{urn},\\ &\mathbf{shake} : \mathbf{urn} \longrightarrow \mathbf{urn},\\ &\mathbf{draw}\ : \mathbf{urn} \longrightarrow \mathbf{bool}\ \}. \end{aligned}$$

Let A and B be the models of $\sum$ given by

$$\begin{aligned} \mathbf{urn}^A &= \text{NAT}, & \mathbf{urn}^B &= \text{NAT},\\ \mathbf{fill}^A(0) &= \text{NAT}, & \mathbf{fill}^B(0) &= \text{NAT},\\ \mathbf{shake}^A &= \lambda n.\{i \mid 0 \le i \le n\}, & \mathbf{shake}^B &= \lambda n.\{i \mid i = 0 \lor i < n\},\\ \mathbf{draw}^A &= \lambda n.\{(n > 0)\}, & \mathbf{draw}^B &= \lambda n.\{(n > 0)\}. \end{aligned}$$

One verifies that A and B are implementation equivalent (cf. Section 4.4). A crucial difference between the models A and B is shown by the following program:

```
P = | [ n := fill(0) ;
        do draw(n) → n := shake(n) od;
        draw(n) ] |.
```

When applied to model A, program P need not terminate. This means that delivery of a result is postponed indefinitely (not that **error** is yielded). When applied to model B, however, the program will terminate and yield the value false.

However, if a user accepts a black box with the specification B, then the user accepts arbitrarily long delay. So, if the box is equipped with model A, the user will never have proof that the box does not function as specified. Therefore we feel justified to consider A as an implementation of B.

5. MORPHISMS OF MODELS AND SUBMODELS

In general, the only direct way to prove that model A is an implementation of model B is by supplying a systematic interpretation M, which assigns to every value $x \in s^A$ a corresponding value $y \in s^B$. Consequently, if p is a string, an element $x \in p^A$ has an image element $y \in p^B$. If f: $p \rightarrow q$ is an arrow, the set of results $f^A(x)$ in q^A is to be mapped into the set of results $f^B(y)$. Since users have complete access to external domains, the interpretation has to be the identity on the external domains. Such systematic interpretations will be called morphisms of models. We shall prove that the existence of a morphism of models M: $A \rightarrow B$ is a sufficient (but not necessary) condition for A to be an implementation of B.

5.1 Specialization, Image, and Composition of Operations

Definition. Let f, g: $X \rightarrow Y$ be operations. We say that f is a *specialization* of g (notation $f \ll g$) if $f(x)$ is a subset of $g(x)$ for every element $x \in X$. One might say that g is more nondeterministic than f (cf. [16]).

The *composition* of operations f: $X \rightarrow Y$ and g: $Y \rightarrow Z$ is defined as the operation $g \circ f$: $X \rightarrow Z$ with $g \circ f(x) = g(f(x))$. Note that $f(x)$ is a nonempty subset of Y^+, so that $g(f(x))$ is a nonempty subset of Z^+ by Section 2.2. Since maps are special cases of operations (cf. Section 2.2), the composition also makes sense in the case that f or g is a map. If both f and g are maps, the composition of the maps f and g is the map corresponding to the composition of the operations f and g. So no ambiguity can arise.

5.2 Morphisms of Models

Let A and B be models of a given signature $\sum$. Let M be a prescription that assigns to every sort $s \in S$ a map s^M: $s^A \rightarrow s^B$. If $p = s_1 \cdots s_m$ is a string of sorts, we define the map between the Cartesian products (cf. Section 2.3), given by

$$p^M(x_1 \cdots x_m) = (y_1 \cdots y_m) \qquad \text{with} \quad y_i = s_i^M(x_i).$$

Definition. The prescription M is said to be a *morphism* of models $M\colon A \to B$ if it satisfies the following two conditions:

(1) If s is an external sort, then $s^M\colon s^K \to s^K$ is the identity map of the external domain s^K.

(2) If $f\colon p \to q$ is an arrow of $\sum$, then we have a specialization of compositions (cf. Section 5.1)

$$q^M \circ f^A \ll f^B \circ p^M\colon p^A \to q^B,$$

where the operations are shown in the following diagram:

$$\begin{tikzcd} p^A \arrow[r, "f^A"] \arrow[d, "p^M"] & q^A \arrow[d, "q^M"] \\ p^B \arrow[r, "f^B"] & q^B \end{tikzcd}$$

If $x \in p^A$ is a value of A, the value $p^M(x) \in p^B$ is called the *M-image* of x in B.

The morphism of models $M\colon A \to B$ is said to be *strict* if in the above condition (2) we always have an equality $q^M \circ f^A = f^B \circ p^M$. A model A is said to be a *submodel* of B if s^A is a subset of s^B for every sort s and if the system of inclusion maps $s^A \to s^B$ defines a morphism of models $A \to B$. It is called a *strict* submodel if the inclusion morphism $A \to B$ is strict.

Remarks. Strict morphisms are called homomorphisms in [6] and [10, p. 228]. Morphisms correspond to the weak homomorphisms mentioned in [6]. The comparison of models with the same domains in [10, p. 94] is a special kind of morphism or submodel relationship. We stress morphisms rather than strict morphisms, as they form a more powerful tool to prove implementation or equivalence of models. See remark (1) of Section 5.4 and the proof of the theorem in Section 7.4. The class of models of a given signature, with morphisms of models as defined here, forms a category (cf. [11]).

5.3 Reachable Values, the Term-Generated Submodel

Definition. Let A be a model of $\sum$. A value $x \in p^A$ is said to be *reachable* if there is an accumulated arrow $f\colon r \to p$ from an external string r, and an external value $w \in r^K$, such that x is in $f^A(w)$. The *term-generated* submodel B of A is defined as follows. If s is a sort, the subset s^B consists of the reachable values of s^A. If $f\colon p \to q$ is an arrow and x is in p^B (cf. Section 2.3), then $f^B(x)$ is defined as the set $f^A(x)$. In fact, one verifies that $f^A(x)$ is contained in q^B (note that p and q are strings of sorts). It follows that B is a strict submodel of A (cf. Section 5.2).

LEMMA. *Let $M\colon A \to B$ be a morphism of models, and let $f\colon p \to q$ be an accumulated arrow (cf. Section 4.2). Then we have*

(a) $q^M \circ f^A \ll f^B \circ p^M\colon p^A \to q^B$;

(b) if f is a rearrangement arrow, then $q^M \circ f^A = f^B \circ p^M$; *and*

(c) if morphism M is strict, then $q^M \circ f^A = f^B \circ p^M$.

SKETCH OF PROOF. Part (b) is easy. The proofs of (a) and (c) consist of straightforward inductions on the complexity of the accumulated arrow f.

THEOREM. *Let $M\colon A \to B$ be a morphism of models of $\sum$.*

(*a*) *Then A is an implementation of B.*
(*b*) *If the morphism M is strict, then A and B are implementation equivalent.*

PROOF. Let $f\colon p \rightarrow q$ be an accumulated arrow between external strings. Then $p^A = p^K$, $q^B = q^K$, and p^M and q^M are the respective identity maps. By the above lemma, we have

$$f^A \ll f^B\colon p^K \rightarrow q^K,$$

with equality $f^A = f^B$ in case (b). □

5.4 Examples of Implementations

Let the signature $\sum$ be given by

$$\begin{aligned} S &= \{\ \mathbf{urn}, \mathbf{bool} := \mathrm{BOOL}\ \}, \\ F &= \{\ \mathbf{fill} \quad : \ \varepsilon \quad \rightarrow \mathbf{urn}, \\ &\qquad \mathbf{draw} : \mathbf{urn} \rightarrow \mathbf{bool}\ \}. \end{aligned}$$

Let A be the model of $\sum$ defined by

$$\begin{aligned} \mathbf{urn}^A &= \mathrm{BOOL}, \\ \mathbf{fill}^A(0) &= \mathrm{BOOL}, \\ \mathbf{draw}^A &= \lambda v.\mathbf{if}\ v \rightarrow \mathrm{BOOL} [\!]\, \neg v \rightarrow \{\mathbf{false}\}\ \mathbf{fi}. \end{aligned}$$

Let B and C be the submodels of A given by

$$\begin{aligned} &\mathbf{urn}^B = \{\mathbf{true}\}, \qquad \mathbf{fill}^B(0) = \{\mathbf{true}\}, \qquad \mathbf{draw}^B = \lambda v.\mathrm{BOOL}; \\ &\mathbf{urn}^C = \mathrm{BOOL}, \qquad \mathbf{fill}^C(0) = \mathrm{BOOL}, \qquad \mathbf{draw}^C = \lambda v.\{v\}. \end{aligned}$$

Since they are submodels of A, models B and C are implementations of A (cf. (a) of the Theorem in Section 5.3). We show that A and B are not implementations of C. In fact, let **copy** be the rearrangement arrow

$$\mathbf{copy} = (\mathbf{urn\ urn}]b[\mathbf{urn})\colon \mathbf{urn} \rightarrow \mathbf{urn\ urn}$$

induced by the map $b\colon \{1, 2\} \rightarrow \{1\}$ with $b = \lambda i.1$. We form the partial compositions (cf. Section 4.2)

$$\begin{aligned} f_2 &= (\mathbf{copy}[\mathbf{urn}]\mathbf{fill}) : \quad \varepsilon \quad \rightarrow \mathbf{urn\ urn}, \\ d_2 &= (\mathbf{draw}[\varepsilon]\mathbf{draw}) : \mathbf{urn\ urn} \rightarrow \mathbf{bool\ bool}, \\ e_2 &= (d_2[\mathbf{urn\ urn}]f_2) : \quad \varepsilon \quad \rightarrow \mathbf{bool\ bool}. \end{aligned}$$

The accumulated arrow e_2 is between external strings. It distinguishes the models A and B from C. In fact, the operations

$$e_2^A,\ e_2^B,\ e_2^C\colon \mathrm{NULL} \rightarrow \mathrm{BOOL} \times \mathrm{BOOL}$$

satisfy

$$e_2^A(0) = e_2^B(0) = \mathrm{BOOL} \times \mathrm{BOOL} \qquad e_2^C(0) = \{(v, v) \mid v \in \mathrm{BOOL}\}.$$

This proves that A and B are not implementations of C.

Models A and B are implementation equivalent. In fact, since B is a submodel of A, it remains to be seen that A is an implementation of B. This is done by constructing a morphism $M\colon A \rightarrow B$. By condition (1) of Section 5.2, a morphism $M\colon A \rightarrow B$ is determined by the map between the internal domains

$$\mathbf{urn}^M\colon \mathbf{urn}^A \rightarrow \mathbf{urn}^B.$$

Since $\mathbf{urn}^B = \{\mathbf{true}\}$, we have to define $\mathbf{urn}^M = \lambda v.\mathbf{true}$. Condition (2) of Section 5 remains to be verified. There are two arrows to consider. One verifies that the arrow symbol **fill** leads to the equality of operations

$$\mathbf{urn}^M \circ \mathbf{fill}^A = \mathbf{fill}^B \circ \varepsilon^M\colon \mathrm{NULL} \to \{\mathbf{true}\}.$$

Similarly, if the identity map of BOOL is denoted by **id**, the symbol **draw** leads to the specialization

$$\mathbf{id} \circ \mathbf{draw}^A \ll \mathbf{draw}^B \circ \mathbf{urn}^M\colon \mathbf{urn}^A \to \mathrm{BOOL}.$$

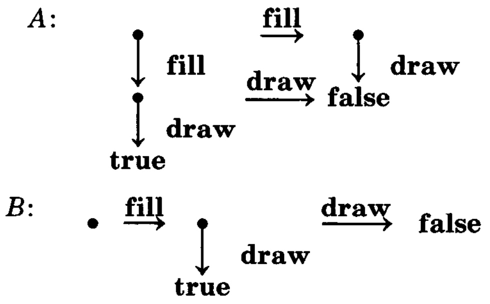
This shows that $M\colon A \to B$ is a morphism of models. Therefore the models A and B are implementation equivalent.

Remarks

(1) Although A and B are implementation equivalent, B is not a strict submodel of A, and the morphism of models M is not strict. So, the implication in (b) of the theorem in Section 5.3 cannot be reversed.

(2) Under the fairness assumptions, models A and B can be distinguished by observations. Let us assume that, in every set of sufficiently many experiments, all possible results of an operation will be found. Then a user can discover that the model A has a certain value $v \in \mathbf{urn}^A$ with $\mathbf{draw}^A(v) = \{\mathbf{false}\}$, whereas B does not have such a value.

(3) Models A and B can be transformed in process graphs (cf. [1]) in the following way:



It is easy to verify that the two graphs are not bisimilar (cf. [1, 1.2.2.1]). In fact, bisimilarity is more closely related to observable equivalence of models, a concept that we do not formally define in this paper.

(4) If, for reasons of memory sharing, users are forbidden to copy internal values, then the model C cannot be distinguished from A and B. This argument leads to a different theory, where the rearrangement maps b of Step 3 of Section 4.2 have to be injective.

5.5 Equivalent Implementations without a Morphism

We construct two deterministic models of the data-type pointer table of Section 3. We forbid overflow by adding the following axiom:

Axiom The operation **position** is total (cf. Section 2.2).

Assume that the set ITEM is countable, so that we may choose an injective mapping

$$\textbf{hash}\colon \text{ITEM} \rightarrow \text{NAT}.$$

It is easy to specify a model A of a pointer table without overflow, which has $\textbf{pointer}^A = \text{NAT}$ and which models a hash table with hash function **hash**. We can also specify a model B with $\textbf{pointer}^B = \text{NAT}$, which models a stack implementation based on an infinite array of items indexed by the set NAT. The models A and B are implementation equivalent. In fact, one can show that all models of a pointer table without overflow are implementation equivalent. However, there does not exist a morphism of models between A and B because there is no adequate correspondence between the sets $\textbf{pointer}^A$ and $\textbf{pointer}^B$.

If C is the model of Section 3.3, based on the set $P = \text{NAT}$, then we have unique morphisms $M\colon A \rightarrow C$ and $N\colon B \rightarrow C$ that induce the identity maps on the set NAT. So the two deterministic models can be compared by means of the nondeterministic model C.

6. TERMINAL MODELS AND SPECIFICATIONS: THE DATA-TYPE BAG

6.1 Complete Determination or a Universal Model

The concept of sufficient completeness of an axiomatization (cf. [5]) has the following analogue in our model-theoretic situation: A nondeterministic data type $(\sum, T)$ can be called *completely determined* if all models A in T are implementation equivalent, and if for every model A in T and every accumulated arrow $f\colon p \rightarrow q$ between external strings the operation $f^A\colon p^A \rightarrow q^A$ is deterministic.

Although this leaves room for unobservable nondeterminism, we prefer not to strive for complete determination. In fact, we prefer a situation where T contains one "universal" model B such that all models $A \in T$ are implementations of B. Then the model B may serve as a specification of any model in T (cf. [9]).

It is an important question of programming methodology how to design and specify relevant classes T. In our experience it helps to look for the existence of a nice universal model B. As soon as B is determined, we prefer to have the class T as wide as feasible.

Definition. The *liberal class* $L(B)$ is defined to consist of all models A that admit some morphism of models $M\colon A \rightarrow B$.

In some cases, the class $L(B)$ turns out to have a nice axiomatization. In principle, the class of all implementations A of B is more important, but usually this class is not tractable. The concept of universality suggested above is technically inconvenient. It lacks uniqueness. In order to express uniqueness, we introduce isomorphisms of models. Then we introduce terminality of models, as a technical version of universality. This concept will be illustrated in the data-type bag. For comparison we also discuss initial models, which leave no room for nondeterminism.

6.2 Isomorphisms of Models

Definition. A morphism of models $M\colon A \rightarrow B$ is called an *isomorphism* if there is a morphism $N\colon B \rightarrow A$ such that the compositions $N \circ M$ and $M \circ N$ are the respective identity morphisms of A and B. Models A and B are said to be *isomorphic* if there exists an isomorphism $M: A \rightarrow B$. It is easy to prove that a morphism of models $M: A \rightarrow B$ is an isomorphism if and only if M is strict and all maps $s^M: s^A \rightarrow s^B$ with $s \in S$ are bijective (cf. [7, 2.6]).

6.3 Terminal Models and Initial Models

Definition. Let $(\sum, T)$ be an NDDT. A model $B \in T$ is said to be *terminal* in T if for every model $A \in T$ there is precisely one morphism of models M: $A \rightarrow B$. A model $C \in T$ is said to be *initial* in T if for every model $A \in T$ there is precisely one morphism $N: C \rightarrow A$. It is well known that, if T has a terminal model B, then this model B is unique up to isomorphism, and similarly for an initial model C (cf. [7, 10.1]).

Remarks. Since all models in T are implementations of B, the model B is universal in the sense of Section 6.1. Some NDDTs have no terminal model. For example, the pointer table of Section 3 does not have a terminal model. This is proved by means of the mathematical models of Section 3.3.

On the other hand, all models of T are implemented by C. So, model C may be useful, if it exists, but leaves no freedom to the implementer. It is a "universal implementation." In fact, if class T contains all submodels of its models, it can be proved that an initial model in T is term generated and deterministic.

6.4 The Nondeterministic Data-Type Bag

Nondeterminism enables us to make a common generalization of stacks, queues, and priority queues. We call it **bag**. Let ITEM be a given external domain. The signature $\sum$ is defined by

$$\begin{aligned} S &= \{\ \mathbf{item} := \text{ITEM},\ \mathbf{bag}\ \},\\ F &= \{\ \mathbf{create} : \quad \varepsilon \quad \rightarrow \mathbf{bag},\\ &\qquad \mathbf{push} \ : \mathbf{item\ bag} \rightarrow \mathbf{bag},\\ &\qquad \mathbf{pop} \ \ : \quad \mathbf{bag} \quad \rightarrow \mathbf{item\ bag}\ \}. \end{aligned}$$

The class T consists of the models A of $\sum$ that satisfy the following axioms:

(1) If $v \in \mathbf{create}^A(0)$, then $v \neq \mathbf{error}$ and $\mathbf{pop}^A(v) = \{\mathbf{error}\}$.

(2) There is an auxiliary function **count**: ITEM × $\mathbf{bag}^A \rightarrow$ NAT, such that every element $v \in \mathbf{bag}^A$ satisfies the following conditions:

(a) The set $|\,v\,| = \{x \in \text{ITEM} \mid \mathbf{count}(x, v) > 0\}$ is **finite**.

(b) If $\mathbf{error} \in \mathbf{pop}^A(v)$ then $|\,v\,|$ is empty.

(c) If $(x, w) \in \mathbf{pop}^A(v)$, then

$$\begin{aligned} \mathbf{count}(x, v) &= \mathbf{count}(x, w) + 1, \text{ and}\\ \mathbf{count}(y, v) &= \mathbf{count}(y, w) \text{ for every } y \neq x. \end{aligned}$$

(d) If $w \in \mathbf{push}^A(x, v)$ is not **error**, then

$$\begin{aligned} \mathbf{count}(x, w) &= \mathbf{count}(x, v) + 1, \text{ and}\\ \mathbf{count}(y, w) &= \mathbf{count}(y, v) \text{ for every } y \neq x. \end{aligned}$$

6.5 The "Universal Bag"

The data type **bag** has a terminal model B, such that the type is the liberal class $L(B)$. The idea of the construction is that the states of the **bag** are characterized by the function count. We let H be the set of the functions f: ITEM → NAT such that the support set

$$|f| = \{x \in \mathrm{ITEM} \mid f(x) > 0\}$$

is finite. Functions f, g in H are added in the obvious way. They can be subtracted in H if the difference is everywhere nonnegative. We use the Kronecker delta function δ: ITEM → H given by

$$\delta = \lambda x.(\lambda y.(\mathbf{if}\ x = y \rightarrow 1 \ [\!]\ x \neq y \rightarrow 0\ \mathbf{fi})).$$

Let B be the model of $\sum$ with $\mathbf{bag}^B = H$ and the operations

$$\begin{aligned}
\mathbf{create}^B(0) &= \{\lambda x.0\},\\
\mathbf{push}^B(x, f) &= \{\mathbf{error}, f + \delta(x)\},\\
\mathbf{pop}^B(f) &= \mathbf{if}\ |f| = \varnothing \rightarrow \{\mathbf{error}\}\\
&\quad\ [\!]\ \ |f| \neq \varnothing \rightarrow \{(x, f - \delta(x)) \mid x \in |f|\}\\
&\quad\ \mathbf{fi}.
\end{aligned}$$

The model B belongs to the class T. In fact, one uses the function **count** = $\lambda(x, f).f(x)$.

THEOREM. *The model B is a terminal model of the class T of Section* 6.4. *The class T is equal to the liberal class $L(B)$.*

Comment. The proof is straightforward. It is important as an illustration of the concept of morphism of models. It is left out for reasons of limitations of space.

7. EXTRACTION EQUIVALENCE OF VALUES

In practical implementations of data types, it is often the case that an abstract value has more than one concrete representation. This happens, for example, in the array implementation of a stack, where the array values beyond the stack pointer are neglected. One may adopt the viewpoint that values are equal if they have the same externally observable properties. For other purposes, however, it may be better to consider such values as different but equivalent values. In this way, one may arrive at the notion of behavioral equivalence. Surprisingly, there are two different formalizations of this notion. In this section we investigate the notion of extraction equivalence (cf. [16, p. 153]). In [10] a finer relation is introduced, under the name of observable equivalence. We come back to that relation in Section 8.

The main purpose of an equivalence relation is that it enables us to treat equivalent values as if they were identical. So, we want to form a quotient model and to prove that this quotient model is implementation equivalent to the original model. This step is not taken in [16]. In this section we develop the adequate theory and give the delicate points of the proof.

7.1 Relations on a Model

Recall that a (binary) relation R on a set X is a subset of the Cartesian product set $X \times X$. The relation R is said to be the identity relation if R consists of the pairs (x, x) with $x \in X$. The relation R is said to be an equivalence relation if it satisfies the following conditions:

(1) If $x \in X$, then $(x, x) \in R$.

(2) If $(x, y) \in R$, then $(y, x) \in R$.

(3) If (x, y) and (y, z) are in R, then (x, z) is in R.

An equivalence relation R on a set X induces a partition of X in equivalence classes

$$x/R = \{y \in X \mid (x, y) \in R\}.$$

We have $x/R = y/R$ if and only if $(x, y) \in R$. The set of the equivalence classes is denoted by

$$X/R = \{x/R \mid x \in X\}.$$

Definition. Let A be a model of a signature $\sum$. A *relation* R on model A is defined to be a prescription that assigns to every sort s a relation s^R on the set s^A. The relation is said to be *internal* if s^R is the identity relation on s^K whenever s is an external sort. The relation is said to be an *equivalence relation* if it is internal and s^R is an equivalence relation on s^A for every sort s.

If R is a relation on a model A, and p is a string of sorts, say, $p = s_1 \cdots s_m$, then two values x, y in p^A are said to be *R-related* if for every index i in $\{1 \cdots m\}$ the pair of the ith components (x_i, y_i) is in s_i^R. The element **error** is said to be R-related to itself, and not to any value of A. Two subsets X and Y of $(p^A)^+$ are said to be R-related if every element of X is R-related to some element of Y and vice versa.

7.2 Extraction Equivalence of Values: Extractable Models

Definition. Let A be a model of $\sum$, and let s be a sort. Values x, y in s^A are said to be *extraction equivalent* if for every accumulated arrow f: $s + p \longrightarrow q$ to an external string q and for every value $z \in p^A$ we have the equality of sets

$$f^A(x, z) = f^A(y, z).$$

Clearly, extraction equivalence is an equivalence relation on model A. Model A is said to be *extractable* if extraction-equivalent values are always identical.

Remark. Extraction equivalence as defined here is a variation of [16, p. 151]. Our accumulated arrows have the possibility to copy internal values and to deliver a string of external values. It is necessary to allow internal probing parameters as the value z in the above definition, in order to obtain results as the lemma below and as the theorem in Section 7.4.

LEMMA. *Let R be the extraction-equivalence relation on a model A. Let $f{:}p \longrightarrow q$ be an accumulated arrow to an external string q. If x, y in p^A are R-related, then $f^A(x) = f^A(y)$.*

PROOF. Assume that $p = s_1 \cdots s_n$. Let x_i and y_i be the respective components of x and y in the domain s_i^A. For every index i, we have $(x_i, y_i) \in s_i^R$. By n applications of the definition, we get

$$f^A(x) = f^A(y_1, x_2, \ldots, x_n) = \cdots = f^A(y_1, \ldots, y_{n-1}, x_n) = f^A(y).$$

Here we use rearrangement to justify the replacement of the ith component instead of the first component. In each case the $n - 1$ other components are used as probing parameters. □

7.3 Construction of the Quotient Model A/R

Let R be an equivalence relation on a model A. We construct the quotient model A/R as follows: For every sort s, we let $s^{A/R} = s^A/s^R$ be the set of the s^R-equivalence classes in s^A (cf. Section 7.1). If s is an external sort, then s^R is the identity relation on s^K. So, in that case we may identify

$$s^{A/R} = s^K/\text{identity} = s^K.$$

Let the s^R-equivalence class of a value x be denoted by $s^M(x) = x/s^R$ in $s^{A/R}$. Then we have a system of maps $s^M\colon s^A \longrightarrow s^{A/R}$. For every string of sorts p, we get an induced map $p^M\colon p^A \longrightarrow p^{A/R}$ (cf. Sections 2.3 and 5.2). Let $f\colon p \longrightarrow q$ be an arrow of $\sum$. Then we define the operation

$$f^{A/R}\colon p^{A/R} \longrightarrow q^{A/R} \qquad \text{by} \quad f^{A/R}(y) = \bigcup \{q^M \circ f^A(x) \mid p^M(x) = y\}.$$

One verifies that A/R is a model of $\sum$ (it is called the *quotient model*), and that $M\colon A \longrightarrow A/R$ is a morphism of models, the *quotient morphism*.

Remark. This construction already occurs in [13]. In our situation, however, we admit **error**, we have many sorts, and we have fixed external domains. For future reference we characterize the quotient model by the following universal property. We omit the proof, which is standard (cf. [7, 8.6]).

PROPOSITION. *Let R be an equivalence relation on a model A. Let M: $A \longrightarrow A/R$ be the quotient morphism. Let $N\colon A \longrightarrow B$ be a morphism of models, such that every pair of R-related values of A has the same N-image in B. Then there is precisely one morphism of models $H\colon A/R \longrightarrow B$ with $N = H \circ M$.*

7.4 Associativity of Partial Compositions: An Auxiliary Result

Consider accumulated arrows

$$\begin{aligned} f &: p_0 \longrightarrow q_0 + r_1 + r_2, \\ g &: r_2 + p_1 \longrightarrow q_1 + r_0, \\ h &: r_0 + r_1 + p_2 \longrightarrow q_2. \end{aligned}$$

Using implicit rearrangements we define the partial compositions

$$\begin{aligned} (g[r_2]f) &: p_0 + p_1 \longrightarrow q_0 + q_1 + r_0 + r_1, \\ (h[r_0]g) &: r_1 + r_2 + p_1 + p_2 \longrightarrow q_1 + q_2, \end{aligned}$$

and the triple compositions

$$\begin{aligned} k &= (h[r_0 + r_1](g[r_2]f)) : p_0 + p_1 + p_2 \longrightarrow q_0 + q_1 + q_2, \\ m &= ((h[r_0]g)[r_1 + r_2]f) : p_0 + p_1 + p_2 \longrightarrow q_0 + q_1 + q_2. \end{aligned}$$

We claim that for every model A of $\sum$ the operations k^A and m^A are equal. In fact, let $x = (x_0, x_1, x_2)$ with $x_i \in p_i^A$, and $y = (y_0, y_1, y_2)$ with $y_i \in q_i^A$. Then y is an element of $k^A(x)$ (and also of $m^A(x)$) if and only if there exists $z = (z_0, z_1, z_2)$ with $z_i \in r_i^A$ such that

$$(y_0, z_1, z_2) \in f^A(x_0) \wedge (y_1, z_0) \in g^A(z_2, x_1) \wedge y_2 \in h^A(z_0, z_1, x_2).$$

The occurrences of **error** in $k^A(x)$ and $m^A(x)$ are also equal. This proves that $k^A = m^A$.

THEOREM. *Let R be extraction equivalence on a model A, and let M: $A \longrightarrow A/R$ be the quotient morphism to the quotient model A/R.*

(*a*) *If f: $p \longrightarrow q$ is an accumulated arrow to an external string q, then $f^{A/R} \circ p^M = f^A$: $p^A \longrightarrow q^K$.*

(*b*) *The models A and A/R are implementation equivalent. The model A/R is extractable.*

PROOF

(a) Since M is a morphism of models and q is an external string, we have the specialization

$$f^A = q^M \circ f^A \ll f^{A/R} \circ p^M\colon p^A \longrightarrow q^K.$$

So it remains to be seen that $f^{A/R} \circ p^M \ll f^A$. This is done by induction on the complexity of the accumulated arrow f. By (b) of the lemma in Section 5.3, rearrangements inside f are harmless. Therefore, by Section 7.4, we may assume that

$$f = (h[r]g)\colon p_0 + p_1 \longrightarrow q_0 + q_1,$$

where g: $p_0 \longrightarrow q_0 + r$ is an ordinary arrow and h: $r + p_1 \longrightarrow q_1$ is an accumulated arrow. By induction we may assume that

$$h^{A/R} \circ (r + p_1)^M = h^A\colon r^A \times p_1^A \longrightarrow q_1^K.$$

Let $x \in p^A$ and $v \in f^{A/R} \circ p^M(x)$. We have to show that v is in $f^A(x)$. Write $x = (x_0, x_1)$ and $v = (v_0, v_1)$ with $x_i \in p_i^A$ and $v_i \in q_i^K$. Put $u = (u_0, u_1)$ with $u_i = p_i^M(x_i)$, so that $v \in f^{A/R}(u)$. By the definition of partial composition, there exists $w \in r^{A/R}$ such that $(v_0, w) \in g^{A/R}(u_0)$ and $v_1 \in h^{A/R}(w, u_1)$. Since g is an ordinary arrow of $\sum$, the set $g^{A/R}(u_0)$ is the union of the M-images of the sets $g^A(x')$ with $p_0^M(x') = u_0$ (cf. Section 7.3). Therefore there exists $x' \in p_0^A$ and $(y, z) \in g^A(x')$ such that the M-images of x', y, z are u_0, v_0, w, respectively. Since q_0 is an external string, we have $y = v_0$. The induction hypothesis implies that

$$h^{A/R}(w, u_1) = h^A(z, x_1).$$

Therefore v_1 is in $h^A(z, x_1)$. This proves that v is in $f^A(x', x_1)$. Having the same M-image u_0, the elements x' and x_0 of p_0^A are R-related. So it follows from the lemma in Section 7.2 that $f^A(x', x_1) = f^A(x_0, x_1)$. This proves that v is in $f^A(x)$, as required. The **error** occurrences are left to the reader.

(b) If $f\colon p \longrightarrow q$ is an accumulated arrow between external strings p and q, then part (a) implies that $f^{A/R} = f^A$. This proves that A and A/R are implementation equivalent. Let s be a sort, and let $u, v \in s^{A/R}$ be extraction equivalent. Choose $x, y \in s^A$ with M-images u, v, respectively. Using part (a) one proves that x and y are extraction equivalent. This implies that

$$u = s^M(x) = s^M(y) = v. \qquad \square$$

7.5 The Extractable, Term-Generated Subquotient

Every model A is implementation equivalent to an extractable and term-generated model. For example, let B be the term-generated submodel of A, and let R denote implementation equivalence on B. Then B/R is extractable and term generated. By the theorems in Sections 5.3 and 7.4, it is implementation equivalent to A. Being a quotient of a submodel, it is called a subquotient.

8. OBSERVABLE EQUIVALENCE OF VALUES

As stated in Section 7, the idea of behavioral equivalence has two different formalizations. In this section we investigate observable equivalence as introduced in [10]. The definition of [10] is indirect and based on mutual recursion. Under an unnecessary hypothesis, the existence of a quotient model is proved. It is postulated that this quotient has the same observable behavior as the original model.

Here we reformulate the definition. We prove that the recursive equation has a solution. It follows from Section 7.3 that a quotient exists. It turns out to be rather easy to show that observable equivalence implies extraction equivalence, and that the quotient model is implementation equivalent to the original model.

Note that observable equivalence closely resembles observation congruence as introduced by Milner for CCS in [12, 7.3]. In the example in Section 10.1.2, it is shown that observable equivalence may differ from extraction equivalence. Roughly speaking, extraction equivalence only looks at direct outputs, whereas observable equivalence also uses previously obtained knowledge about internal values.

8.1 Kapur's Definition of Observable Equivalence

In [10, p. 89], two values of a model A are said to be *observably equivalent* if they are not distinguishable. The latter concept is defined by mutual recursion, as follows:

(a) If s is a sort, values $x, y \in s^A$ are *distinguishable* if s is external and $x \neq y$, or if there exists an accumulated arrow $f\colon s + p \longrightarrow q$ and a value $z \in p^A$ such that the sets $f^A(x, z)$ and $f^A(y, z)$ are distinguishable.

(b) If q is a string of sorts, say, $q = s_1 \cdots s_n$, then values $u, v \in (q^A)^+$ are *distinguishable* if either element is **error** and the other element is not, or if there is an index $i \in \{1 \cdots n\}$ such that the components u_i and v_i are distinguishable in s_i^A. Subsets U and V of $(q^A)^+$ are *distinguishable* if either set contains an element w that is distinguishable from all elements of the other set.

Remarks

(1) We have rephrased the definition of *loc.cit.* in our terms. The use of accumulated arrows is inessential, but convenient. It allows the assumption that in (a) the critical values x, y are in the first component. A probing parameter z is clearly necessary, but I cannot decide whether *loc.cit.* [10] allows it.

(2) The observability can be rather impractical. If the operations of A have unbounded nondeterminism, it may require transfinite induction to distinguish distinguishable values.

8.2 Definition and Construction of Observable Equivalence

We transform the definition in Section 8.1 into a direct definition of observable equivalence itself. In order to express the negations of (a) and (b), we introduce the concept of the derived relation:

Definition. If R is an internal relation on a model A, the *derived relation* DR is defined as follows: If s is a sort, then s^{DR} consists of the pairs $(x, y) \in s^A \times s^A$ such that for every accumulated arrow f: $s + p \longrightarrow q$ and every value $z \in p^A$ the subsets $f^A(x, z)$ and $f^A(y, z)$ of $(q^A)^+$ are R-related (cf. Section 7.1).

One verifies that DR is an internal relation and that DR is contained in R. If R is an equivalence relation on A, then so is DR. In Section 8.1 the sets of distinguishable pairs of values are the smallest sets such that conditions (a) and (b) hold. Therefore the sets of observably equivalent pairs are the largest sets such that the negations of (a) and (b) hold. Rephrasing this we get that the observable equivalence relation should be the largest internal relation R that satisfies $R = DR$.

In order to prove that indeed this equation has a unique largest solution, we consider the collection Φ of all equivalence relations Q on A such that Q contains every internal relation R' on A with $R' = DR'$. The collection Φ is nonempty. In fact, let P be the internal relation such that s^P is the identity relation on s^K for every external sort s, and that $s^P = s^A \times s^A$ otherwise. Then P is an equivalence relation on A, which contains all internal relations on A. So P is in Φ.

Now we define R as the intersection of all members Q of Φ. As every member Q is an equivalence relation that contains all solutions R', the intersection R is also an equivalence relation that contains all solutions R'. It follows that R is in Φ. By the definition of D, we get that DR is in Φ, so that R is contained in DR. This implies that $R = DR$. As R contains all solutions, it is the largest solution. Moreover, R is an equivalence relation.

Definition. Therefore we define *observable equivalence* to be relation R.

8.3 Congruences

Definition. An equivalence relation R on a model A is said to be a *congruence* (cf. [10, p. 27]) (or an ideal, cf. [13, p. 329]) if for every arrow f: $p \longrightarrow q$ and every pair of R-related values $x, y \in p^A$ the sets $f^A(x)$ and $f^A(y)$ are R-related in $(q^A)^+$ (cf. Section 7.1).

Using the arguments of the lemma in Section 7.2, one can see that an equivalence relation R on A is a congruence if and only if $R = DR$. It follows that observational equivalence is a congruence and that it is the largest congruence. The first part of this assertion is not claimed in [10, p. 44].

The theoretical importance of congruences is based on the following fact: Let R be an equivalence relation on A. Then the quotient morphism $M: A \longrightarrow A/R$ is strict if and only if R is a congruence (cf. [13, Theorem 1] or [10, p. 228]).

PROPOSITION. *Let A be a model, and let Q and R be extraction equivalence and observable equivalence, respectively.*

(*a*) *R is contained in Q.*

(*b*) *If A is deterministic, then $R = Q$.*

PROOF

(a) In terms of Section 8.2, we have $Q = DP$. Since P contains R, the derivation Q contains $DR = R$.

(b) Let $f: p \longrightarrow q$ be an arrow, and let $x, y \in p^A$ be extraction equivalent. By the determinism we have unique resulting elements $f^A(x)$ and $f^A(y)$. These elements are easily seen to be extraction equivalent. Therefore Q is a congruence. Since R is the largest congruence, it follows that $Q = R$. □

Definition. A model A is said to be *observable* if observable equivalence is the identity relation. We have the following analogue of the theorem in Section 7.4:

THEOREM. *Let R be the observable-equivalence relation on A. Then A and A/R are implementation equivalent. The model A/R is observable.*

PROOF. Since R is a congruence, the quotient morphism from A to A/R is strict. So A and A/R are implementation equivalent by (b) of the theorem in Section 5.3. The proof of the observability of A/R is straightforward (cf. [7, 8.4]). □

9. INSEPARABLE VALUES AND CONSISTENCY

The main difficulty of the behavior of nondeterministic models is that a model has the freedom to show its abilities, as well as to hide them. In Section 8.1 two values are called distinguishable if the model is able to reveal a difference. It may be, however, that the model is not forced to reveal a difference. In that case the values may be considered as inseparable, or consistent. Actually, we choose these two words to represent two different formalizations of the same intuitive idea. In this section we develop the theory. Later, in Section 10.1.4, it is shown that inseparable values need not be consistent in the technical sense.

The theoretical motivation for this section is that recent studies have shown relationships between terminality of a model A and the possibility to separate its internal values by observations (cf. [9, p. 106] and [17, p. 12]). Note that [9] and [17] only consider deterministic data types. As the condition of being terminal in an arbitrary class of models cannot imply much, we restrict ourselves to the biggest available class, the liberal class $L(A)$ (cf. Section 6.1). We prove that model A is terminal in class $L(A)$ if and only if consistent values in A are always equal.

9.1 Inseparability of Values

Definition. Let A be a model of $\sum$. If s is a sort, let values $x, y \in s^A$ be called *inseparable* if for every accumulated arrow $f: s + p \longrightarrow q$, where p and q are external strings, and for every external value $z \in p^K$ the sets $f^A(x, z)$ and $f^A(y, z)$ are not disjoint.

It is easy to see that extraction equivalence implies inseparability. If the model A is deterministic and term generated, then extraction equivalence and separability are equivalent. In general, inseparability is an internal relation on A (cf. Section 7.1), which need not be an equivalence relation. Unfortunately, inseparability is not sufficiently strong to characterize terminality of the model A. We need the slightly stronger concept of consistency.

9.2 Consistent Relations and Consistent Values

Definition. A relation R on A is said to be *consistent* if it is internal and for every arrow $f: p \longrightarrow q$ and every pair of R-related values $x, y \in p^A$ there exist elements $u \in f^A(x)$ and $v \in f^A(y)$ such that u, v are R-related.

Consistent relations form a generalization of congruences. In fact, if R is a congruence, then R is a consistent equivalence relation. Conversely, if A is a deterministic model and R is a consistent equivalence relation, then R is a congruence.

Definition. If s is a sort, values $x, y \in s^A$ are said to be *consistent* if there exists a consistent relation R on A such that (x, y) is in s^R. Clearly, observably equivalent values are always consistent.

9.3 Another Characterization of Consistent Relations

Let B be a second model of $\sum$, and let $M, N: B \longrightarrow A$ be two morphisms of models. If s is a sort, let s^R be the set of the pairs $(s^M(b), s^N(b))$ in $s^A \times s^A$ with $b \in s^B$. Since M and N induce the identity maps on the external domains, the prescription R is an internal relation on A (cf. Section 7.1).

Definition. This relation R is called the relation *spanned by* the triple (B, M, N).

PROPOSITION. *A relation R on A is consistent if and only if it is spanned by some triple (B, M, N).*

PROOF. First, assume that R is spanned by the triple (B, M, N). Let $f: p \longrightarrow q$ be an arrow of $\sum$, and let $x, y \in p^A$ be R-related. Assume that $p = s_1 \cdots s_m$. Let x_i and y_i be the respective components of x and y in s_i^A. Since R is spanned by (B, M, N), we get an element $b = (b_1 \cdots b_m) \in p^B$ with $x = p^M(b)$ and $y = p^N(b)$. Choose an element $c \in f^B(b)$. Put $u = q^M(c)$ and $v = q^N(c)$. Since M and N are morphisms, we have $u \in f^A(x)$ and $v \in f^A(y)$. If $c \neq$ **error**, then u and v are R-related. Otherwise both elements are equal to **error** and therefore R-related. This proves that R is consistent.

Conversely, assume that R is consistent. We construct a model B as follows: If s is a sort, put $s^B = s^R$. If s is an external sort, then s^B is the identity relation on s^K, so that it can be identified with s^K by means of the diagonal map $\lambda x.(x, x)$. If

p is a string of sorts, the product set p^B is identified with the subset of $p^A \times p^A$ that consists of the R-related pairs (x, y). The element **error** of $(p^B)^+$ is identified with (**error**, **error**). If f: $p \longrightarrow q$ is an arrow of $\sum$ and $z \in p^B$ is identified with the pair (x, y), then $f^B(z)$ is defined to consist of the elements $w = (u, v)$ in the product set $f^A(x) \times f^A(y)$, such that u, v are R-related. By the definition of consistency, this set $f^B(z)$ is nonempty. Therefore B is a model of $\sum$. If s is a sort, let s^M, s^N: $s^B \longrightarrow s^A$ be the projections on the first and the second components, respectively. In this way we get morphisms M, N: $B \longrightarrow A$ such that R is spanned by the triple (B, M, N). □

COROLLARY. *If values $x, y \in s^A$ are consistent, then they are inseparable.*

PROOF. Choose a consistent relation R on A with $(x, y) \in s^R$. Let R be spanned by the triple (B, M, N). Choose $b \in s^B$ with $x = s^M(b)$ and $y = s^N(b)$. Let f: $s + p \longrightarrow q$ be an accumulated arrow with p and q external strings. Let $z \in p^K$. By the lemma in Section 5.3, the nonempty set $f^B(b, z)$ is contained in $f^A(x, z)$ and also in $f^A(y, z)$. Therefore these sets are not disjoint. □

THEOREM 9.1. *A model A of $\sum$ is terminal in its liberal class $L(A)$ if and only if consistent values in A are always equal.*

PROOF. Assume that A is not terminal in $L(A)$. Then there is a model B with two different morphisms M, N: $B \longrightarrow A$. Since M and N are different, there is a sort s and a value $b \in s^B$ such that $s^M(b) \neq s^N(b)$. By the proposition above, these values of A are consistent.

Conversely, assume that $x, y \in s^A$ are different consistent values. By the proposition above, there is a triple (B, M, N) and a value $b \in s^B$ such that $x = s^M(b)$ and $y = s^N(b)$. Since $x \neq y$ the morphisms M and N are different. This implies that A is not terminal in $L(A)$. □

Example. See the theorem in Section 6.5.

THEOREM 9.2. *Let A be a model of $\sum$ that is terminal in its liberal class $L(A)$. Let B in $L(A)$, and let R be a consistent equivalence relation on B. Then the quotient model B/R belongs to $L(A)$.*

PROOF. Let R be spanned by the triple (C, M, N). Since A is terminal in $L(A)$, we have a unique morphism P: $B \longrightarrow A$, and the two compositions $P \circ M, P \circ N$: $C \longrightarrow A$ are equal. It follows that every two R-equivalent values of B have equal P-images in A. So, by the proposition in Section 7.3, there is a unique morphism H: $B/R \longrightarrow A$. □

Remark. In particular, if R is observable equivalence, the observable quotient B/R belongs to $L(A)$. See also the remark in Section 10.1.3.

10. COMPARISON OF SEPARATION CRITERIA

10.1 Summary of Results

In this section we review the main separation concepts of Sections 7–9. In particular, we investigate the implication relations between these concepts. In the cases where no implication has been derived, we give examples to show that no implication holds.

We have four separation criteria on values x, y of a model A:

C0: x, y are observably equivalent (cf. Section 8.1 and [10]).
C1: x, y are extraction equivalent (cf. Section 7.2 and [16]).
C2: x, y are consistent (cf. Section 9.2).
C3: x, y are inseparable (cf. Section 9.1).

$$\begin{tikzcd}
C0 \arrow[r, Rightarrow] \arrow[d, Rightarrow] & C1 \arrow[d, Rightarrow] \\
C2 \arrow[r, Rightarrow] & C3
\end{tikzcd}$$

These four conditions satisfy the above square of implications (cf. the proposition in Section 8.3, Sections 9.1 and 9.2, and the corollary in Section 9.3). If model A is deterministic and term generated, all four conditions are equivalent (by the proposition in Section 8.3, and Section 9.2, we have that C1 implies C0 and that C3 implies C1).

We may also consider the four conditions on A that require that the respective relation reduces to the identity. These conditions are as follows:

D0: A is observable (cf. the second definition in Section 8.3).
D1: A is extractable (cf. Section 7.2).
D2: A is terminal in the class $L(A)$ (cf. Theorem 9.1).
D3: Inseparable values in A are identical.

Clearly, these conditions satisfy the square of implications with the inverse directions:

$$\begin{tikzcd}
D0 & D1 \arrow[l, Rightarrow] \\
D2 \arrow[u, Rightarrow] & D3 \arrow[l, Rightarrow] \arrow[u, Rightarrow]
\end{tikzcd}$$

In order to show that there are not more implications than obtained, we shall give four examples:

(10.1.1) D1 $\not\Rightarrow$ D2: an extractable model A that is not terminal in $L(A)$;
(10.1.2) D0 $\not\Rightarrow$ (D1 or D2): an observable model A, not extractable and not terminal in $L(A)$;
(10.1.3) D2 $\not\Rightarrow$ D1: a not extractable model A, terminal in $L(A)$; and
(10.1.4) (D1 and D2) $\not\Rightarrow$ D3: an extractable model A, terminal in $L(A)$, with different inseparable values.

All examples consist of a term-generated model based on a focused signature and involving a high degree of nondeterminism. They also show that there do not exist more implications between the separation conditions C0, C1, C2, and C3.

10.1.1 *An Extractable Model A that Is Not Terminal in L(A).* We use the signature $\sum$ and the model A of Section 5.4. The two values of $\mathbf{urn}^A$ are not extraction equivalent, but they are consistent. Therefore A is extractable, and not terminal in $L(A)$. The model C of Section 3.3 is a more extensive example.

10.1.2 *An Observable Model A, Not Extractable, and Not Terminal in L(A).* We use a variation of Section 5.4:

$$\begin{array}{l}S = \{\ \mathbf{chaos},\ \mathbf{urn},\ \mathbf{bool} := \mathrm{BOOL}\},\\ F = \{\ \begin{array}{lcl}\mathbf{prepare}: & \varepsilon & \rightarrow \mathbf{chaos},\\ \mathbf{fill} & :\mathbf{chaos} & \rightarrow \mathbf{urn},\\ \mathbf{draw} & :\ \mathbf{urn} & \rightarrow \mathbf{bool}\ \}.\end{array}\end{array}$$

The model *A* is defined by

$$\begin{aligned}\mathbf{chaos}^A &= \mathbf{prepare}^A(0) = \mathbf{urn}^A = \mathrm{BOOL},\\ \mathbf{fill}^A &= \lambda v.\{v,\ \mathbf{true}\},\\ \mathbf{draw}^A &= \lambda v.\{v,\ \mathbf{false}\}.\end{aligned}$$

We have the following diagram:

$$\begin{tikzcd}
\Sigma : & \varepsilon \arrow[r, "\mathbf{prepare}"] & \mathbf{chaos} \arrow[r, "\mathbf{fill}"] & \mathbf{urn} \arrow[r, "\mathbf{draw}"] & \mathbf{bool} \\
A : & 0 \arrow[r] \arrow[dr] & \mathbf{true} \arrow[r] & \mathbf{true} \arrow[r] \arrow[dr] & \mathbf{true} \\
 & & \mathbf{false} \arrow[r] \arrow[ur] & \mathbf{false} \arrow[r] & \mathbf{false}
\end{tikzcd}$$

The two elements of $\mathbf{urn}^A$ can be distinguished. Therefore the elements of $\mathbf{chaos}^A$ can be distinguished. This shows that *A* is observable. The two elements of $\mathbf{chaos}^A$ are extraction equivalent. So *A* is not extractable. The elements of $\mathbf{urn}^A$ and also of $\mathbf{chaos}^A$ are consistent. So *A* is not terminal in *L*(*A*).

10.1.3 *A Not Extractable Model A, Terminal in L(A).* The signature is a variation of the one of Section 10.1.2. The drawing operator has an extra integer parameter.

$$\begin{array}{l}S = \{\ \mathbf{chaos},\ \mathbf{urn},\ \mathbf{int} := \mathrm{INT},\ \mathbf{bool} := \mathrm{BOOL}\},\\ F = \{\ \begin{array}{lcl}\mathbf{prepare}: & \varepsilon & \rightarrow \mathbf{chaos},\\ \mathbf{fill} & :\ \mathbf{chaos} & \rightarrow \mathbf{urn},\\ \mathbf{draw} & :\mathbf{urn}\ \mathbf{int} & \rightarrow \mathbf{bool}\ \}.\end{array}\end{array}$$

The model *A* is specified by

$$\begin{aligned}\mathbf{chaos}^A &= \mathrm{BOOL},\\ \mathbf{prepare}^A(0) &= \mathrm{BOOL},\\ \mathbf{urn}^A &= \mathrm{BOOL} \times \mathrm{INT},\\ \mathbf{fill}^A &= \lambda v.\{v\} \times \mathrm{INT},\\ \mathbf{draw}^A &= \lambda((v, m), n).\ \begin{array}[t]{l}\mathbf{if}\ m > n \rightarrow \mathrm{BOOL}\\ [\!]\ m = n\ \ \rightarrow \{v\}\\ [\!]\ m < n\ \ \rightarrow \{\neg v\}\\ \mathbf{fi}.\end{array}\end{aligned}$$

The two elements of $\mathbf{chaos}^A$ are extraction equivalent. In fact, given an accumulated arrow *f*: **chaos** + $p \rightarrow q$ where *q* is an external string, and a value $z \in p^A$, the **fill** operation can always choose a sufficiently large second component *m* so that the **draw** operation has all freedom. On the other hand, consistent values are identical. In fact, let *R* be a consistent relation on *A*. Then *R*-related values in $\mathbf{urn}^A$ are identical, and therefore the same holds in $\mathbf{chaos}^A$. It follows that A is terminal in $L(A)$.

Remark. If Q is the extraction equivalence on this model A, then the quotient model A/Q does not belong to the liberal class $L(A)$. Compare with Theorem 9.2.

10.1.4 *An Extractable Model B, Terminal in L(B), with Different Inseparable Values.* We use the signature of Section 10.1.3, and a model B that has the same domains as the model A of Section 10.1.3 and also the same operations **prepare** and **fill**. The only modification is

$$\begin{aligned}\mathbf{draw}^B = \lambda((v, m), n).\mathbf{if}\ & m = n \rightarrow \{v\} \\ [\!]\ & m \neq n \ \ \rightarrow \{\mathbf{error}\} \\ \mathbf{fi}.&\end{aligned}$$

It is easily verified that B is extractable and that every consistent relation on B is contained in the identity relation. So B is terminal in $L(B)$. The two elements of $\mathbf{chaos}^B$ are inseparable, since every accumulated arrow that distinguishes these two values has **error** as a common result.

10.2 Bounded Nondeterminism

The examples in Sections 10.1.3 and 10.1.4 make essential use of unbounded nondeterminism. I have not been able to decide whether similar examples exist with only bounded nondeterminism.

11. CONCLUDING REMARKS

11.1 Results

We have shown that, if one is prepared to separate a data type from its syntactic specification, nondeterminism can be described adequately and rigorously by means of the calculus of sets and set-valued mappings. We obtained a very general semantical formalism, which can probably be restricted in many different ways, depending on the application at hand.

One of the results is that data types whose operations modify more than one argument or return multiple values need not be excluded from theoretical investigation. In fact, the calculus of accumulated arrows has nothing to do with nondeterminism, but may be useful in general (i.e., also in the special case of deterministic data types).

We introduced morphisms in order to compare different models of the same signature. A morphism enables us to relate the values from the models in a systematic way. This is necessary for discussion of equivalence relations, degrees of nondeterminism, and universality properties of models. Morphisms can be used to prove that one model implements another model, but a morphic relationship is not necessary for an implementation. The correctness criterion for implementations (Section 4.3) leads to a preference for extraction equivalence over the more usual observable equivalence. In fact, extraction equivalence is the coarser relation, but nevertheless leads to an implementation equivalent quotient model (cf. the theorem in Section 7.4).

11.2 Suggestions

We have specified two useful NDDTs (in Sections 3.1 and 6.4), but although [7] and [10] give indications, a systematic investigation of specification methods for NDDTs remains a subject of future research.

In order to deal with hierarchical nondeterministic data types, one could extend our concept of signature to include external operations. Then an extractable model of one signature could provide the external domains and operations of a new signature.

It may be useful to construct accumulated arrows with arbitrary control structures (cf. Section 4.5). That would enable formal descriptions of vertical implementations (cf. (1) of Section 1.2).

It seems that our theory can be applied to concurrency. In fact, partial composition of accumulated arrows might have been designed for that very purpose. One may want to withdraw the condition of Section 2.2 that the result sets of an operation are nonempty. That would require reexamination of our proofs, but most of the results would remain valid.

REFERENCES

1. BERGSTRA, J. A., AND KLOP, J. W. Algebra of communicating processes. In *Proceedings of the CWI Symposium on Mathematics and Computer Science* (Amsterdam, Oct. 6–7, 1986). Centre for Mathematics and Computer Science, Amsterdam, 1986, pp. 89–138.
2. BERZTISS, A. T., AND THATTE, S. Specification and implementation of abstract data types. In *Advances in Computers*, vol. 22. M. C. Yovits, Ed. Academic Press, New York, 1983, pp. 295–353.
3. DIJKSTRA, E. W. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, N.J., 1976.
4. GOGUEN, J., AND MESEGUER, J. Universal realization, persistent interconnection and implementation of abstract modules. In *Automata, Languages and Programming (ICALP 82). Lecture Notes in Computer Science*, vol. 140, M. Nielson and E. M. Schmidt, Eds. Springer-Verlag, New York, pp. 265–281.
5. GUTTAG, J. V., AND HORNING, J. J. The algebraic specification of abstract data types. In *Programming Methodology*, D. Gries, Ed. Springer-Verlag, New York, pp. 282–308.
6. HANSOUL, G. E. A subdirect decomposition theorem for multialgebras. *Algebra universalis 16* (1983), 275–281.
7. HESSELINK, W. H. A theory of non-deterministic data structures. Computing Science Notes, Rep. CS 8503, Dept. of Mathematics and Computer Science, Groningen State Univ., Groningen, 1985.
8. JONES, C. B. Development methods for computer programs including a notion of interference. Ph.D. dissertation, Rep. PRG 25, Oxford Univ., 1981.
9. KAMIN, S. Final data types and their specification. *ACM Trans. Program. Lang. Syst. 5*, 1 (Jan. 1983), 97–123.
10. KAPUR, D. Towards a theory for abstract data types. Ph.D. thesis, Tech. Rep. MIT/LCS/TR-237, MIT, Cambridge, Mass., 1980.
11. MACLANE, S. *Categories for the Working Mathematician.* Springer-Verlag, New York, 1971.
12. MILNER, R. *A Calculus of Communicating Systems, LNCS* 92. Springer-Verlag, New York, 1980.
13. PICKETT H. E. Homomorphisms and subalgebras of multialgebras. *Pac. J. Math. 212*, (1967), 327–342.
14. SANELLA, D., AND WIRSING, M. A kernel language for algebraic specification and implementation (extended abstract). In *Foundations of Computation Theory. Lecture Notes in Computer Science*, vol. 158, M. Karpinski, Ed. Springer-Verlag, New York, 1983, pp. 413–427.
15. STOY, J. E. *Denotational Semantics: The Scott–Strachey Approach to Programming Language Theory.* MIT Press, Cambridge, Mass., 1977.

16. Subrahmanyam, P. A. Nondeterminism in abstract data types. In *Automata, Languages and Programming. Lecture Notes in Computer Science*, vol. 115, S. Even and O. Kariv, Eds. Springer-Verlag, New York, 1981, pp. 148–164.
17. Wirsing, M., Pepper, P., Partsch, H., Dosch, W., and Broy, M. On hierarchies of abstract data types. *Acta Inf. 20* (1983), 1–33.
18. Wirth, N. *Algorithms + Data Structures = Programs*. Prentice-Hall, Englewood Cliffs, N.J., 1976.

Received January 1986; revised October 1986; accepted December 1986