Nondeterministic Operators in Algebraic Frameworks

Sigurd Meldal *Michal Walicki*

Technical report CSL–TR–95–664
Program Analysis and Verification Group Report No. 69

March 1995

Computer Systems Laboratory
Department of Electrical Egineering and Computer Science
Stanford University
Stanford, CA 94305–4055

Abstract

A major motivating force behind research into abstract data types and algebraic specifications is the realization that software in general and types in particular should be described ("specified") in an abstract manner. The objective is to give specifications at some level of abstraction: on the one hand leaving open decisions regarding further refinement and on the other allowing for substitutivity of modules as long as they satisfy a particular specification.

The use of nondeterministic operators is an appropriate and useful abstraction tool, and more: nondeterminism is a *natural* abstraction concept whenever there is a hidden state or other components of a system description which are, methodologically, conceptually or technically, inaccessible at a particular level of abstraction.

In this report we explore the various approaches to dealing with nondeterminism within the framework of algebraic specifications. The basic concepts involved in the study of nondeterminism are introduced. The main alternatives for the interpretation of nondeterministic operations, homomorphisms between nondeterministic structures and equivalence of nondeterministic terms are sketched, and we discuss various proposals for, respectively, the initial and terminal semantics. We make some comments on the continuous semantics of nondeterminism and the problem of solving recursive equations over signatures with binary nondeterministic choice. And we then go on to present the attempts at reducing reasoning about nondeterminism to reasoning in first order logic, and gives an example of a calculus dealing directly with nondeterministic terms. Finally, rewriting with nondeterminism is discussed: primarily as a means of reasoning, but also as a means of assigning operational semantics to nondeterministic specifications.

Key Words and Phrases: Algebraic specifications, nondeterminism, formal methods, software design, foundations of computer science, programming language semantics.

CONTENTS

Introduction

Mathematics never saw much of a reason to deal with something called "nondeterminism." It works with values, functions, sets, relations. In computing science, on the other hand, nondeterminism has been an issue from the very beginning, if only in the form of nondeterministic Turing machines or nondeterministic finite state machines. Early references to nondeterminism in computer science go back to the sixties [38, 84]. A great variety of theories and formalisms dealing with it have been developed during the last two decades. There are the denotational models based on power domains [103, 117, 51, 106], the predicate transformers for the choice construct [29, 30, 104, 118], modifications of the λ-calculus [73, 4, 49]. Nondeterminism arises in a natural way when discussing concurrency, and models of concurrency typically also model nondeterminism. There are numerous variants of process languages and algebras [13, 89, 90, 50, 54, 52, 65, 39, 11], event structures [136, 135, 134, 2], state transition systems [83, 71], Petri nets [100, 109].

In terms of modeling nondeterminism may be considered a purely operational notion. However, one of the main reasons for considering nondeterminism in computer science is the need for abstraction, allowing one to disregard irrelevant aspects of actual computations. Typically, we prefer to work with models which do not include all the details of the physical environment of computations such as timing, temperature, representation on hardware, etc. Since we do not want to model all these complex dependencies, we may instead represent them by nondeterministic choices. The nondeterminism of concurrent systems usually arises as an abstraction of time. Similarly nondeterminism is also a means of increasing the level of abstraction when describing sequential programs [85, 130], and as a way of indicating a "don't care" attitude as to which among a number of computational paths will actually be utilized in a particular evaluation [30].

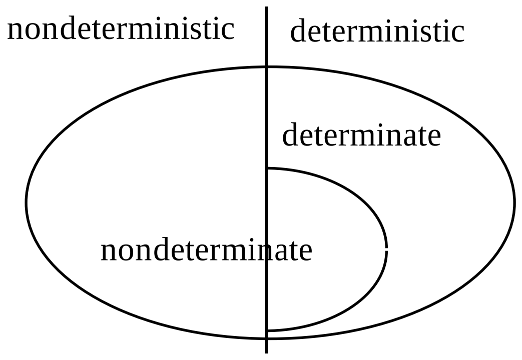
The variety of approaches referred to above indicates the possible difficulties in gathering all the pieces into one uniform theory, let alone a short presentation. In this paper we are concerned with algebraic specifications and hence will consider only a part of the whole picture of nondeterminism. As far as the basic notions related to nondeterminism and the associated algebraic formalisms are concerned, the paper is by and large self-contained. Only occasionally references to other areas presupposing some prior knowledge will be made.

In section 1 the basic concepts involved in the study of nondeterminism are introduced. Sections 2-5 discuss the semantic issues, and 6-7 reasoning with nondeterminism. The main alternatives for the interpretation of nondeterministic operations, homomorphisms between nondeterministic structures, and equivalence of nondeterministic terms are sketched in 2. Sections 3 and 4 discuss various proposals for, respectively, the initial and the terminal semantics. In section 5 we make some comments on the continuous semantics of nondeterminism and the problem of solving recursive equations over signatures with binary nondeterministic choice. 6 presents the attempts at reducing reasoning about nondeterminism to reasoning in first order logic and then gives an example of a calculus dealing directly with nondeterministic terms. In 7 rewriting with nondeterminism is discussed: primarily, as a means of reasoning, but also as a means of assigning operational semantics to nondeterministic specifications.

1. Basic concepts

In this section we present informal definitions of the basic concepts and distinctions involved in the study of nondeterminism.

Nondeterminism and nondeterminacy



Roughly speaking, nondeterminacy concerns *syntax*, nondeterminism *semantics*. The constructs which always yield unique result are *determinate*, those which may yield different results when invoked several times *nondeterminate*. The presence of a nondeterminate construct in an expression does not force the corresponding operation to be nondeterministic. Determinacy implies determinism but nondeterminacy does not necessarily imply nondeterminism and, as observed in [18], the problem whether a nondeterminate term is deterministic or not is, in general, undecidable. We are primarily concerned with the intentional nondeterminism originating from the presence in the language of some constructs which have nondeterministic semantics.

Nondeterminism and underspecification

When developing a software system in a number of refinement steps, we are often interested in specifying the functionality of the system uniquely but only with respect to some relevant properties. That is, each model of the specification is a standard (deterministic) structure but we do not identify one unique model. We then speak of underspecification. Later in the development process we may add more properties, whenever we find it appropriate, and so restrict the model class. Thus underspecification functions also, like nondeterminism, as a means of abstraction. It bears a resemblance to nondeterminism in that it leaves open the possibility of choosing among several admissible models. The important difference between the two notions may be roughly expressed thus: underspecification admits a choice between different models but nondeterminism admits choices within one model. While underspecification fits into the concepts of classical logic and model theory smoothly, the treatment of nondeterminism leads to complications and, typically, requires introduction of non-standard features both into the models and logic. For this reason some researchers postulate the use of underspecification as the primary, if not the only, means of abstraction. Others consider it insufficient and try to design formalisms which capture the phenomenon of nondeterminism as distinct from underspecification.

Representational vs. “real” nondeterminism

There are essentially two reasons why one might want to include the concept of nondeterminism in the traditional algebraic specification methods:

(1) *Real* nondeterminism.

The system being specified really is nondeterministic – its behavior is not fully predictable, nor fully reproducible, no matter how detailed our knowledge of its initial state.

(2) *Representational* (or *pseudo*-) nondeterminism [64, 121, 130].

The behavior of the system being specified may be fully predictable in its final implementation (i.e. deterministic), but it may not be so at the level of abstraction of the specification.

Though many think of representational nondeterminism as identical to underspecification, they turn out to be technically and conceptually quite distinct (as we shall see shortly).

Whether the world *really* is nondeterministic or not we leave to the physicists and philosophers to ponder. A computer system *in isolation* certainly is deterministic: When started from a particular state (given in full detail) twice, both executions will demonstrate identical behavior. Possible sources of perceived nondeterminism lie only in the unpredictability of the environment such as hardware failures or human factors. Considering all such factors as parts of the total state given in full detail may obviate the perceived nondeterminism, but leads to undesirable complexity and is possible only in principle.

The primary argument in favor of accepting nondeterministic operators is instrumental, and identical to the credo of the abstract data type community: One should specify a system *only in such detail that any implementation satisfying the specification also satisfies the user, and no more.* It turns out that nondeterministic operators ease the process of specifying systems by allowing one to disregard irrelevant aspects – be they the external influences or implementation details – and thus reducing the danger of overspecification resulting from technical rather than methodical reasons.

For purposes of discussion it may be convenient to further identify three variants of representational nondeterminism: (1) abstraction from hidden state, (2) abstraction from time, and (3) abstraction from external entities. Though these may be dealt with uniformly, they have often been considered distinct. In particular, the introduction of nondeterminism as a result of abstraction from *time* is usually taken as a given in the process algebra community without thereby necessarily accepting abstraction over *state* as requiring nondeterminism for specification purposes.

How does this use of nondeterminism differ from the usual notion of underspecification? Consider for a moment a user-defined choice function $\sqcup$ from sets of integers to integers, returning one of the elements of the set:

For instance, $\sqcup(\{0,1\})$ may return either of the values 0 and 1. If choice were just an underspecified function, then we would have that $\sqcup(\{0,1\})=\sqcup(\{1,0\})$, since the arguments of the function are equal (though not syntactically identical) in the two terms. In practical terms, this would require the choice operator always to return the same value when applied to a particular set. I.e., $\sqcup(\{0,1\})$ is always 0, or always 1.

However, this kind of underspecification does not allow for abstraction from (conceptually) invisible entities that might influence the choice (such as a hidden state, timing or interaction with a human being). E.g., if set values were implemented as unordered sequences with new elements always added to the front of the sequence, this underspecified description of the choice function would disallow using a simple implementation of choice as picking the first element of the sequence, since such an implementation would sometimes return the value 0, sometimes the value 1, when applied to the set {0,1}, depending on which of the two elements were added first. If we were to treat choice as a nondeterministic operator, on the other hand, then such a straightforward implementation (though deterministic) would be quite acceptable, both formally and according to the usual intuition about the requirements of an operator picking some element from a set [107, 132].

Similarly, if the implementation of the choice function asked a human operator to pick an element then one would encounter the same difficulty: The behavior of human beings may be deterministic, but even were that the case their inner state determining that behavior is not available for inspection. A specification needs to abstract from that inner state, and nondeterminism is the right concept for doing that.

And similarly again, if the choice depended upon timing properties (e.g. the set was distributed among a number of processors, and the choice function simply queried them all, returning the first (in terms of time) value returned to it by one of these processors) the abstraction from timing properties would introduce a seeming nondeterminism.

The implementation relation also arises in the distinction between *loose* and *tight* relationships between structures modeling nondeterministic operators (see below).

Bounded and unbounded nondeterminism

Bounded nondeterminism refers to the case where every terminating computation has only a finite number of possible results; *unbounded* – to the one where the set of the possible results may be infinite.

It may be argued [30, 54] whether unbounded nondeterminism has a plausible computational interpretation. A deterministic program *P* which takes a natural number *n* as input and outputs a natural number *m* corresponds to a partial recursive function. The relation

$$R_P(n,m) \Leftrightarrow P(n)=m$$

is recursively enumerable (RE), and the subset L_P of natural numbers for which *P* does not terminate is co-recursively enumerable. For nondeterministic program *P*, the input-output relation $R_P(n,m)$ is also RE, but when the involved nondeterminism is unbounded L_P is much more complex than co-RE. Firstly, (non-)termination cannot be guaranteed and L_P is the set of those inputs on which *P may* not terminate. Then it is shown in [25] that for erratic (see below) unbounded nondeterminism L_P has the complexity Σ^1_1. This either presents a serious challenge to the Church-Turing thesis and the classical notion of computability or, perhaps, discredits unbounded nondeterminism as a notion without computational relevance. Nevertheless, even if one might agree that the notion of unbounded nondeterminism is not feasible from the implementation point of view, it may even so provide an invaluable abstraction mechanism at the specification level.

Unbounded nondeterminism creates severe difficulties in models based on fixed point semantics because, unlike bounded nondeterminism, it is not continuous in the standard constructions of power domains. However, it should be observed that noncontinuity is not caused by the nondeterminism, but rather by the unboundedness. Noncontinuity may also arise in a determinate language, for instance, in a language admitting quantification over infinite sets.

Biased agents

The paradigmatic concept of nondeterminism is probably that of arbitrary choice. Choosing nondeterministically between *a* and *b* there is a possibility that we get *a* and also a possibility that we get *b*. The choice may be influenced by additional factors which nevertheless leave it, to some extent, undetermined. If this is the case we may speak of the (*agent* making the) choice being *biased*.

The most extensively studied case of such agents involves bias with respect to possible termination. *Erratic* choice is completely arbitrary – it may happen that such a choice will lead to a nonterminating computation and it may happen that it will eventually produce some result. *Angelic* choice, on the other hand, will always avoid branches which may lead to nontermination. If there is a computational path leading to a successful result, angelic nondeterminism will guarantee that such a path will be found. Finally, a *demonic* choice will do the opposite and always follow the path, if such exists, leading to a nonterminating computation.

In terms of implementation, erratic nondeterminism is the least problematic. A choice may be performed locally without consideration of the possible consequences of the choices made. It may be also thought of as an unpredictable environment beyond the control of the program. An operational intuition of angelic nondeterminism [25] can be a system which, whenever a choice is to be made, spawns several new processes, one for each among the possible results of the choice. The first process to terminate causes all other processes to stop as well. The demonic case can be analogous except that *every* process must terminate if the whole computation is to terminate. Demonic nondeterminism is in [21] called *backtrack* nondeterminism – it is there thought of as a system which makes a choice and follows the path until it terminates. If it does, the system backtracks to the point at which the last choice was made and follows another path. It will terminate only after having checked that all possible choices lead to terminating computations.

The terms erratic, angelic, demonic are used not only to refer to the termination aspect but, generally, to the situations where nondeterminism involves arbitrary choices where the results are either "desirable" or "undesirable," e.g., with respect to definedness.

The first version of angelic nondeterminism [84] was related to this kind of preference for "desirable" behavior. It was local (like the erratic one) in the sense that it chose among its arguments looking only at their values rather than at the consequences of its own choice. If both arguments were well defined (completed computations) any of them might have been chosen erratically but if any of the arguments was undefined the other one was chosen.

This is an example of the *value* bias [110] where every agent may have its own preference as to which values to choose. Such a bias can be modeled by a partial order on the values expressing the preferences. An agent presented with the choice between a and b first considers whether any of the values is to be preferred (is greater in the ordering), in which case this value is selected. Otherwise the choice is arbitrary. For instance, the ordering for the angelic agent in the last paragraph would be the *flat* partial order: $a \leq b$ iff $a=\bot \vee a=b$.

Another form of a bias is *fairness*. If an erratic agent has infinitely many occasions to choose between a and b it may happen that it will always choose a. A fair agent will eventually choose also b.[1] There is a close connection between fair choice and unbounded nondeterminism as the well-known example illustrates [30, 99, 3]:

```
$b$:=$\mathbb{T}$ ; $x$:=0;
while $b$ do
   $b$:=$\mathbb{F}$ $\sqcup$ $x$:=$x$+1                                        (Prog1)
od;
```

If $\sqcup$ denotes erratic (demonic) choice this program may (will) not terminate. Any number may be returned by a terminating (erratic) computation. If $\sqcup$ is fair (in this case it means also angelic) then the program will *always* terminate. There is no upper bound on the number of iterations before this will happen, though, so any natural number may be returned as the value of x. This means that a "solution" to the fairness problem would also provide a mechanism to implement unbounded nondeterminism.

Many subtleties arise in this connection: the distinction between strong termination (which requires an upper bound on the number of iterations) vs. weak (where no such bound exists) [5], tight implementation (which produces all the results prescribed by the specification) vs. loose (which only needs to produce some of them) [21, 99] and, of course, various kinds of fairness ranging from the "ideal" fairness as in the example above to forms of computable fairness where the bound on the delay in selecting any of the alternatives is determined by some computable function. The latter is shown in [25] to have complexity Σ^0_1, i.e., essentially the same as functions computable by deterministic programs. (Thus fairness will restrict the functions "computable" by a nondeterministic program.)

As a final example of biased agents we mention *probabilistic* nondeterminism [68, 108, 111, 81]. Here every choice is made with some probability distribution P which may depend only on the values (e.g., whenever a choice between a and b is to be made, $P(a)=1/3$ and

[1] This is known as *unbounded* fairness. There are many different notions of fairness but we do not focus on them here.

$P(b)=2/3$), or also on the agents making the choice (so that $P_M(a,b)$ for an agent M may differ from $P_N(a,b)$ for another agent N.) Of course, the formalism for the description, as well as the models, of probabilistic nondeterminism are considerably more complex than the formalisms and models of non-probabilistic nondeterminism. They are probably more appealing to the community interested in probabilistic algorithms than to workers in the field of formal specifications and abstract data types.

Singular and plural

In deterministic programming the distinction between call-by-value and call-by-name semantics is well-known. The former corresponds to the situation where the actual parameters to function calls are evaluated and passed as values. The latter allows also parameters which are function expressions, passed by a kind of Algol copy rule [113], and which are evaluated only when a need for their value arises.

The nondeterministic counterparts of these two notions are what [110] calls *singular* and *plural* semantics of parameter passing. Other, very closely related distinctions go under the names call-time-choice vs. run-time-choice [26, 49, 50], inside-out (IO) vs. outside-in (OI) [34, 35]. The different names reflect slight differences in meaning of the concepts. Except where we are entering into a more detailed discussion of this distinction (7) we will adopt the terminology [110] (taking the risk of abusing it a bit for the sake of a more intuitive exposition).

The terminology of [110] reflects the meaning of the nondeterminate terms as representing sets of possible results. Evaluation of such a term yields a unique result, hence when evaluation of the argument is required at the moment of the call it represents a single value. When a term is passed by some variant of the textual copy rule and several evaluations of it in the body of the operation can happen independently of each other, then we can picture the situation as passing the whole set of possible results where each reference to the parameter name picks (independently) one among the possible results.

Etc. ...

Among other, more particular distinctions which may be occasionally referred to, we have:

Weak vs. *strong* [110]. Weak nondeterminism means that, although some internal parts of a computation may happen nondeterministically, the eventual result is uniquely determined by the input. For instance, confluent rewriting of a term may apply different rules (chosen nondeterministically) but will always arrive at the same normal form; a term may be called weakly nondeterministic if it is nondeterminate and deterministic. Strong nondeterminism is, of course, the nondeterminism proper.

Tight vs. *loose* [99, 21, 110]. This distinction concerns the relation between two, possibly nondeterministic, structures. If M is one of them and displays some nondeterminism then N is said to be tight if it displays exactly the same amount of nondeterminism, and it is loose if its nondeterminism is, possibly, more restricted than of M. For instance, if M is a specification then, typically, its implementation N will be allowed to decrease the nondeterminism of some operations (loose). Similarly, if M is a programming language with nondeterminate constructs, then we may think of a (loose) deterministic implementation N of M as being correct if every operation in N returns a result which is among the possible results of the corresponding operation in M.

Restrained vs. *unrestrained* [110]. The former is nondeterminism of single programming constructs – “choose arbitrary number”, and the latter is nondeterminism allowing choice of different execution paths – “goto label1 or label2”. The latter can be easily implemented using the former. Nevertheless, the names come from the fact that denotational models of the latter are much more complex and require power domain construction over function spaces with a non-flat ordering.

Notation

A specification SP is a pair (Σ, Π) of the signature and the formulae in some language which will depend on the context. The set of ground terms over Σ will be denoted W_Σ, and $W_{\Sigma,X}$ will denote terms with variables from the set X. Terms may be determinate or nondeterminate – when speaking of their interpretation we will say "*function*" whenever a deterministic operation is meant, and "*operation*" whenever the term may denote a nondeterministic operation. For $t \in W_{\Sigma,X}$, we will let $\{t\}$ denote the set of its possible results, possibly with a superscript, as in $\{\}$, for the structure A in which t is interpreted.

Nondeterministic choice is denoted by "$\sqcup$". Sometimes $\sqcup$ will denote binary choice, written $x \sqcup y$, but usually its argument will be a set – either because the profile of $\sqcup$ in Σ declares it so, or because the appropriate axioms of commutativity, associativity and idempotency are given. For the set-valued operator the notation $\sqcup.\{x,y\}$ will be used.

Equality, as a primitive of the language, is written in the infix notation, $x=y$; $=(x,y)$ is used to denote equality as a defined predicate. The symbol "$\triangleq$" indicates syntactic identity of its arguments.

An upper case letter such as "A" usually denotes a set or a model with carrier $|A|$. (The latter notation is also occasionally used for the cardinality of a set. We expect no confusion arise from this overloading of notation.) $\mathcal{P}(A)$ is the power set of A, $\mathcal{P}^+(A)$ the set of its nonempty subsets. $\mathcal{P}A$ will denote (some variant of) power set structure. Instead of $S_1 \times \ldots \times S_i$ we will write S^i.

2. Semantic preliminaries

The algebraic approach to nondeterminism is dominated by the use of power set structures. We use the name "power *set* structure" (algebra, model) as the generic description of most algebraic models of nondeterminism. In particular, *power algebras* are just a special case of power *set* algebras. We include here also the *function oriented* and *relational* constructions since they are closely related to what one would naturally associate with the expression "power set structure". However, there is no standard definition of this notion and the choices one has to make are not merely esthetical or technical. In this section we sketch the main alternatives of modeling operations and homomorphisms using power set structures. The following definition will be used extensively:

Definition 2.1. We say that a function $f: \mathcal{P}(A) \to \mathcal{P}(B)$ is

1. *additive*, $f \in [\mathcal{P}(A) \to_{\cup} \mathcal{P}(B)]$, iff $\forall S \in \mathcal{P}(A) : f(S) = \bigcup \{f(\{s\}) \mid s \in S\}$,
2. *strict*, $f \in [\mathcal{P}(A) \to_{+} \mathcal{P}(B)]$, iff $f(\varnothing) = \varnothing$, and
3. *preserves singletons*, $f \in [\mathcal{P}(A) \to_{1} \mathcal{P}(B)]$, iff $\forall S \in \mathcal{P}(A) : |S|=1 \Rightarrow |f(S)|=1$.

Functions satisfying both 1. and 2. will be indicated by $\to_{\uplus}$.

□

2.1. Operations ...

The carrier of a power set algebra $\mathcal{P}A$ is (usually) the power set of some (*underlying* or *basis*) set A: for each sort $S \in \mathrm{S}$, the carrier of S is $\mathcal{P}(\mathrm{S}^A)$, or $\mathcal{P}^+(\mathrm{S}^A)$. (In the notation we usually drop the sort indexing – it is always implicitly present.) The elements of A will be called *individuals* – interpretation of terms in $\mathcal{P}A$ is usually based on the interpretation of variables as individuals (singular semantics [110]). The interpretation of a nondeterminate operation $f: S^i \to S$ offers several choices.

Functional models: $f^A = \{f_z: (S^i)^A \to S^A\}$

Here the carrier is the set A rather than its power set. Every f is interpreted as a set $\{f_z\}$ of deterministic functions. Operationally it may mean: whenever f is to be applied, it first chooses some i and then applies the (deterministic) function f_i, i.e., all nondeterminism is resolved at the beginning of the computation.

A possible argument against the functional model is that it is not fully abstract. Instead of looking at the input-output relation which gives the abstract view of a program (specification) it looks inside it and distinguishes models with the same observable behavior, [59, 58]:

Example 2.2

Let A and B be two models of the operation f: S→S:

$$S^A = \{0,1\} = S^B$$

$$\begin{array}{ll} f^A = \{f_0,f_1\} & f^B = \{f_2,f_3\} \\ f_0(0) = f_0(1) = 0 & f_2(0) = 0,\ f_2(1) = 1 \\ f_1(0) = f_1(1) = 1 & f_3(0) = 1,\ f_3(1) = 0 \end{array}$$

For any input, f^A and f^B can return the same result, and hence might be called indistinguishable. The above semantics, however, would claim that A and B are different because both computations are performed in different ways.

□

This vice may become a virtue if the way in which operations are computed matters. It is easy to abstract such a structure of computations and look only at the result sets produced by a nondeterministic operation [126].

The model reflects also the fact that each computation of any program produces a unique result – observing the results produced in *one* execution of a program does not supply sufficient information to decide whether we run a nondeterministic f, which happened to choose an f_i, or whether we run a deterministic f_i. Thus

$$m_i \in f(n) \ \Leftrightarrow\ \exists f_i{\in}f : f_i(n){=}m_i \tag{2.3}$$

In this view there is, in general, no deterministic program equivalent to a nondeterministic one. Both produce unique results in every computation on a given input, but all computations of the former produce the same result, while different computations of the latter can produce different results. In terms of the automata theory this amounts to viewing each nondeterministic finite state machine NM as a set of deterministic machines D_z, each accepting exactly one string from the language of NM.

This functional approach has received relatively little attention in the literature. The domains of indexed sets used in [9] instead of power domains remind one of indexed functions. A more elaborate investigation of the functional models is reported in [126] where the relationships to the next kind of models are studied as well. We will denote functional models by *FMod*.

Multialgebras: f^A: $(S^i)^A \rightarrow \mathcal{P}(S^A)$

This is the most common approach [102, 64, 53, 58, 93, 8]. The arguments to the operations are individuals and the result is the set of possible outcomes. Thus operations are modeled as deterministic set-valued functions.

This view corresponds to the equivalence of nondeterministic and deterministic (finite state or Turing) machines. The central point of this equivalence is the definition of languages accepted by the former in which nondeterminism is eliminated: a nondeterministic machine NM accepts a string s, $s{\in}\mathcal{L}$(NM), iff there exists an accepting computation starting on s. Writing S for the initial state, $\mathbb{Y}$ for a final (accepting) state, and ε for the empty string, we have

$$s{\in}\mathcal{L}(\text{NM}) \ \Leftrightarrow\ \exists \text{ computation } C : (S,s) \rightarrowtail^{C} (\mathbb{Y},\varepsilon)$$

The existential quantifier eliminates all nondeterminism – it does not matter any more which computation is performed; an accepting computation C either exists or not, and the language of NM is uniquely defined. Hence, an equivalent deterministic machine DM can be constructed by "dovetailing" all possible computations of NM. Analogously, if f is a (nondeterministic) multioperation, the result set $f(n)$ can be computed deterministically by evaluating ("dovetailing") all possible computation paths, as it is done for the Turing machines.

Composition of multioperations is defined using the following simple fact [34, 106]:

Proposition 2.4. For every f: $A \to \mathcal{P}(B)$ there exists a unique $f^{\mathcal{P}}$: $\mathcal{P}(A) \to_{\uplus} \mathcal{P}(B)$ (strict, additive) such that the following diagram commutes:

$$\begin{tikzcd} & A \arrow[dl, "\{\_\}"'] \arrow[dr, "f"] & \\ \mathcal{P}(A) \arrow[rr, "f^{\mathcal{P}}"] & & \mathcal{P}(B) \end{tikzcd}$$

□

{_} denotes the canonical embedding sending every element to the singleton set. Thus composition is defined as: $g(f(x)) = g^{\mathcal{P}}(f(x)) = \bigcup\{g(e) \mid e \in f(x)\}$. Also here it is natural to interpret the carrier as the set A rather than its power set, but the canonical embedding {_} and additivity of the operations make the transition between the two very easy. We will denote multimodels by *MMod*.

Here the two models from example 2.2 would be identified, since both f^A and f^B applied to any argument return the set {0,1}.

In most cases, one lets an operation f map A to $\mathcal{P}^+(B)$ rather than $\mathcal{P}(B)$. The former corresponds to the *total* while the latter to the *partial* models in which $f(a)=\emptyset$ indicates that f is undefined on a. With the above definition of composition this partial interpretation implies angelic nondeterminism.

Example 2.5

Let the sort N = {0,1,2} and S={a,b,c}, and the operations f: N→S, g: S→N be such that

$f(0) = \{a,b\}$ $\quad$ $g(a) = \{0,1\}$

$f(1) = \{a,c\}$ $\quad$ $g(b) = \{2\}$

$f(2) = \emptyset$ $\quad$ $g(c) = \emptyset$

Then

$g(f(0)) = \{0,1,2\}$

$g(f(1)) = g^{\mathcal{P}}(f(1)) = g^{\mathcal{P}}(\{a,c\}) = \{0,1\} \cup \emptyset = \{0,1\}$

$g(f(2)) = \emptyset$

□

To obtain more flexibility in modeling biased nondeterminism with multialgebras some additional constructions in the specification language are needed, analogous to those for partiality in the deterministic case, e.g., the bottom element $\perp$ [77, 63, 91], or definedness predicate [21, 18, 59, 58].

Although the classes MMod and FMod are not isomorphic there is a strong sense of correspondence between the two:

Proposition 2.6 [127]. Every functional algebra F determines a multialgebra M and vice versa.

Proof:

Let F be a functional algebra where $f^F = \{f_1, f_2, ...\}$. Define the multialgebra M by:

- $|M| = |F|$
- $f^M(x) = \{f_i(x) | f_i \in f^F\}$.

The reverse implication is analogous, though a bit more technical and not constructive:

- $|F| = |M|$

For every f and $m \in |F|$ (of appropriate sort), let $\kappa_{f,m}$ denote the cardinality of the set $f^M(m)$, and let $\kappa_f = \cup \kappa_{f,m}$. Then, for every m there is a surjection

$\alpha_m: \kappa_f \to f^M(m)^m$

and we define

- $f^F = \{f_i \mid i \leq \kappa_f\}$ where, for every m, $f_i(m) = \alpha_m(i)$.

(Note that F isn't unique and the axiom of choice is needed to actually determine it.)

□

Treatment of partiality in a functional model will be quite different from that illustrated in example 2.5. Unless we introduce a $\perp$ element (or a definedness predicate) explicitly, there will be no default interpretation of undefinedness (such as the empty set) in the carrier of a functional model.

Other consequences of the multialgebraic interpretation can best be explained by contrasting it with the third possibility:

Power algebras: $f^A: \mathcal{P}(S_1^A) \times ... \times \mathcal{P}(S_i^A) \to \mathcal{P}(S^A)$

Here every function takes a set as an argument and returns a set as the result. There are still two possible ways to interpret this:

a) Every element of the result set is a possible outcome of applying the operation to *any* element of the argument set. Under this interpretation a power algebra is just a more concise expression of a multialgebra, as the following straightforward consequence of the proposition 2.4 shows ($[A \to B]$ denotes the partial order of functions from A to B ordered pointwise):

Proposition 2.7. $[A \to \mathcal{P}(B)] \simeq [\mathcal{P}(A) \to_{\uplus} \mathcal{P}(B)]$.

b) The other (and from now on the only) meaning is that every element of the result set is a possible outcome of applying the operation to the argument *set*. Without any additional conditions such a definition begs the whole question of nondeterminism because what we obtain is a deterministic structure which just happens to have a power set as the carrier. In particular, there is no distinction between elements, or 1-element sets, and other sets. An operation f may, for instance, be such that $f(\{0,1,2\}) = \{0\}$ and $f(\{0\}) = \{0,1\}$. This is counterintuitive [33, 103] because one expects that an increase in the nondeterminism of the arguments to an operation should not result in a decreased nondeterminism of the result.

To meet the intuitive understanding one would require that the operations in a power algebra be $\subseteq$-monotonic. $\subseteq$-monotonicity does not imply additivity, and so proposition 2.7 does not yield an isomorphism with multialgebras. The possibility of non-additive functions between power sets offers new possibilities. If we aim at plural semantics of parameter passing [110, 130] we are forced to allow the arguments of functions to be sets (and to let variables refer to sets rather than individuals).

Example 2.8

Let f be the operation

$f(x) = \text{if } x=x \text{ then } 0 \text{ else } 1.$

In any multimodel (x will refer to an individual and) the result set of f will be $\{0\}$ for all x. In particular, $\{f(a\sqcup b)\}=\{0\}$. Some authors [77, 53] focus on the purely semantic issues, i.e., do not consider any specification language. But by adopting the multimodel of operations they are forced to adopt the singular semantics of the operation symbols.
If, on the other hand, we take f as mapping sets to sets, we obtain

$f(a\sqcup b)$ = if $a\sqcup b=a\sqcup b$ then 0 else 1,

which may give $\{f(a\sqcup b)\}=\{0,1\}$.
□

Actually, the last equality does not follow by itself in a power model, but depends further on the interpretation of the equality $a\sqcup b=a\sqcup b$ (which may be interpreted as element- or set-equality). We discuss this further in 2.3 and 7. The singular semantics can be obtained, according to 2.7, if we impose the additional restriction of additivity. Thus power algebras, but not multialgebras, give us the possibility to define both singular and plural semantics.

The reasons why this approach is relatively unpopular in spite of its apparent generality are probably mostly pragmatic and similar to the reasons why call-by-name semantics has been superseded by the call-by-value in the deterministic setting (besides the fact that additive function spaces have nicer formal properties). It may be argued that programs written in terms of call-by-value are significantly clearer and more tractable than those using call-by-name. It should be also observed that one risks obtaining only uncountable models where most elements are unreachable when defining the carrier as a set of all (also infinite if the basis set is infinite) subsets.

Power models will be denoted by *PMod*.

Relational models: $f^A \subseteq S_1{}^A\times\ldots\times S_i{}^A\times S^A$
Although relational structures are well known in the universal algebra [27, 80, 79] they have not quite found their way into the world of algebraic specifications, where the intuition of functional application and its result plays the central role. Use of relations in semantic definitions is made in [99, 7, 93, 25, 18], and, in a categorical setting, in [119]. In so far as input-output behavior is concerned the relational models are isomorphic to multimodels.

Proposition 2.9. $[A\rightarrow\mathcal{P}(B)] \simeq [A \rightarrow(B\rightarrow\text{Bool})] \simeq [A\times B \rightarrow \text{Bool}] \simeq \mathcal{P}(A\times B)$.

With the relational product as the definition of composition, one obtains angelic nondeterminism as with multialgebras (example 2.5). The most typical use of relations is for describing termination properties and this is how they are used in [99, 7, 93, 25, 18]. One introduces a pair of relations: one for defining the input-output relation for the terminating computations, and one for characterizing the inputs which may (will) lead to divergence.

At the level of specifications, [121, 64, 21, 18] pointed in the direction of the relational structures by describing nondeterministic operations by means of *characteristic predicates*. But the relations are used as auxiliary definitions of the semantics and are not fully integrated into the formalism of the specification language. None of the above works developed a relational specification language. An exception is the work from [1, 55, 123] which attempts to develop a theory of data types based on the notion of a relation instead of a function. The relational language leads to concise, albeit hard to read, specifications and gives powerful support in performing calculations. Since nondeterminism is implicit in the notion of a relation – functions being just a special case of relations – the relational approach offers a uniform treatment of deterministic and nondeterministic operations.

2.2. ... homomorphisms ...

Homomorphisms for multialgebras were defined already in [101,102], and then in [45, 53, 59, 64, 93].

Recall that a homomorphism φ between (deterministic) structures A and B is a family of mappings φ_S: $S^A \to S^B$ for every $S \in \mathrm{S}$ such that the following diagram commutes for every $f \in \mathbf{F}$:

$$\begin{tikzcd}
(S^i)^A \arrow[r, "f^A"] \arrow[d, "\varphi_{S_i}"'] & S^A \arrow[d, "\varphi_S"] \\
(S^i)^B \arrow[r, "l"] & S^B
\end{tikzcd}$$

i.e., 1. for every constant c: $\to \mathrm{S}$ $\varphi_S(c^A) = c^B$, and

 2. for every operation f: $\mathrm{S}_i \to \mathrm{S}$ $\varphi_S(f^A(x_i)) = \mathrm{f}^B(\varphi_{S_i}(x_i))$ for all $x_i \in \mathrm{S}_i^A$

The transition to nondeterministic structures again introduces several possibilities of generalization. They are only loosely related to the choice of interpretation of the operations. One general remark applies to all of them: Since f is set-valued the result of following the leftmost path in the diagram gives a set $\{f^B(\varphi(a))\}$. Similarly, $\{f^A(a)\}$ is a set and hence the result of applying φ to it (all its members) will give a set. The two basic possibilities are therefore:

tight homomorphism: $\{\varphi(f^A(a))\} = \{f^B(\varphi(a))\}$ (2.10)

loose homomorphism: $\{\varphi(f^A(a))\} \subseteq \{f^B(\varphi(a))\}$ (2.11)

Any of these two conditions may replace the homomorphism condition 1.-2. Loose homomorphisms correspond to nonincreasing nondeterminism in the pre-image. Notice that this does not preclude the cardinality of the set $\{f^A(a)\}$ being greater than this of $\{f^B(\varphi(a))\}$. Then some values produced by $f^A(a)$ must be equivalent under φ. Loose homomorphisms (or their equivalents) are often used as the implementation criteria since it is generally accepted that one should allow deterministic (less nondeterministic) implementations of nondeterministic data types [121, 64, 94, 53, 132].

Both kinds of homomorphisms are used in the literature, though often under different names. The vocabulary becomes even more confused since many authors introduce the partiality considerations into the definitions. ([59, 18]. See [21, 93, 94] for more detailed and idiosyncratic notions.)

Element homomorphisms: φ: A$\to$B.

This is the most common way of defining homomorphisms in a nondeterministic context, and we will denote it by *EHom*. Here the basic entities are individuals and homomorphisms send individuals to individuals. If multialgebras or power algebras are involved one still may use this notion of homomorphism since pointwise extension then defines the mapping between the corresponding power sets. In either case the homomorphism condition will be modified to (2.10) or (2.11).

Multihomomorphisms: φ: A$\to\mathcal{P}^+$(B).

In [59] the element homomorphisms are generalized to the set-valued ones. There is at least one advantage to be gained from this. In the deterministic case the initial structure for a given signature Σ is the collection of all words, W_Σ. The interpretation of Σ in a structure A is given by the (unique) homomorphism $\mathcal{I}$: $W_\Sigma \to A$. In the nondeterministic case we may want to interpret some terms as sets. The notion of multihomomorphisms makes such an interpretation a homomorphism whereas element homomorphisms do not.

$\mathcal{I}$ may be a homomorphism in EHom if we do not explicitly distinguish individuals from sets. Then, if a structure B happens to be a power algebra, the mapping $\mathcal{I}$: $W_\Sigma \to B$ will actually send terms to sets since here B is the power set of some set. But then sets, which are the interpretations of terms, cannot be identified with the *result* sets since the distinction between individuals and sets disappears (sets are individuals in $\mathcal{P}(B)$).

Multihomomorphism will be denoted by *MHom*.

Power homomorphisms: $\varphi: \mathcal{P}(A) \to \mathcal{P}(B)$.

This notion may lead to the peculiarities reminiscent of those of unrestricted (to $\subseteq$-monotonic) functions in power algebras. The intention behind the definition of a homomorphism is to make the mappings from individuals to individuals and from sets to sets "compatible", even if the specification language is insufficient for dealing with this distinction. Both EHom and MHom ensure such "compatibility" – mapping from a power set is obtained by pointwise extension of the mapping from individuals.

Example 2.12

Suppose that Σ contains only two constants of sort S, 0 and 1. Let A and B be the following power algebras:

$A=\mathcal{P}\{\underline{0},\underline{a},\underline{b}\}$ $\quad$ $B=\mathcal{P}\{\underline{0},\underline{1}\}$

$0^A=\{\underline{0}\}$, $1^A=\{\underline{a},\underline{b}\}$ $\quad$ $0^B=\{\underline{0}\}$, $1^B=\{\underline{1}\}$

A power homomorphism $\varphi: A \to B$ must send $\{\underline{0}\}$ to $\{\underline{0}\}$ and $\{\underline{a},\underline{b}\}$ to $\{\underline{1}\}$. The rest is arbitrary so, for instance, we may have $\varphi(\{\underline{a}\})=\varphi(\{\underline{b}\})=\{\underline{0}\}$.

□

This does not look very plausible. Again, as in the case of the power functions, it helps a lot if we insist that homomorphisms be $\subseteq$-monotonic. But the point is how such a requirement is expressed. If we just restrict the legal morphisms to those which are $\subseteq$-monotonic then we will exclude mappings which, like the one above, preserve the Σ-structure and satisfy the compatibility condition defining homomorphisms.

It can be seen from the example that the trouble arises from the fact that we do not have a syntactic operation which would correspond to the semantic operation of set construction. Choice is not really such a constructor. If interpreted as set union, it would enable us to construct only the set $\{\underline{0},\underline{a},\underline{b}\}$ in A but not, for instance $\{\underline{a}\}$. Thus, instead of extending the gap between syntax and semantics we might consider extending the specification language with an appropriate operation (predicate) such that the homomorphism condition wrt. to this operation would imply $\subseteq$-monotonicity. Several works [77, 59, 87, 91] introduce such an operation. If, in addition, the language contains a predicate expressing that something is an individual [91] then homomorphisms are again determined by the images of singletons. This leads to the same class of mappings as EHom since strict, additive mappings from $\mathcal{P}(A)$ to $\mathcal{P}(B)$ which, in addition, preserve singletons are isomorphic to the mappings from A to B:

Proposition 2.13. $[A \to B] \simeq [\mathcal{P}(A) \to_{\uplus,1} \mathcal{P}(B)]$.

See also [102, 106] for the results concerning the relationship between various forms of mappings between power sets. Extending the notational analogy with the models we will denote the power homomorphisms by *PHom*.

2.3. ... and equivalencies.

Although it might seem natural that = should be interpreted in power set structures as set equality (since the operations return sets) it is not obvious that the natural choice is the better.

Element equality

First of all, the sets returned by the operations represent *possible* results. But each particular application of every operation will return only *one unique* result. Thus, for instance, if

$\{s\}=\{0,1\}$ and $\{t\}=\{0,1\}$ it may very well happen that one application of s returns 0 while an application of t returns 1. One may require that two equal terms *always* return the same results. This *element equality* reflects a strict view of observability – two terms are equal only if they are *necessarily* equal, *i.e.* they are deterministic and always return identical results. This notion of equality corresponds to the function oriented model and, like that model itself, has not received much attention in the literature (except for [127, 133, 130]). Its apparent oddity lies in the fact that, since a nondeterministic operation may return different results at different invocations, it is not even an equivalence relation. This reflects the lack of referential transparency which is intrinsic to nondeterministic operators.

Set equality

Slight variations of the set equality have been applied by many authors: under the name of *ideal congruence* in [102], *extraction* equivalence in [53, 121], *observational* equivalence in [64]. Variations concern mainly whether one uses plain set equality or whether the relation is defined in the context of observability. In the latter case there are hidden and visible sorts, and terms s and t are extraction equivalent iff sets returned by all visible contexts $C[\_]$ when applied to s and t are equal.[1] The simplest definition says that an equivalence relation E is *extraction* equivalence iff

$$\forall (t,s)\in E,\ \forall C[\_]\in W_{\Sigma,\{x\}},\ \forall a\in\{C[t]\}\ \exists b\in\{C[s]\} : (a,b)\in E \tag{2.14}$$

The following result is then reported in [53, 102]:

Theorem 2.15. Let SP be a specification, M a multimodel, E an extraction equivalence on M, and $[x]$ the E-equivalence class of $x\in M$. Then M/E is a multimodel, where

- the carrier of M/E is the set of E-equivalence classes $\{\ [x] \mid x\in|M|\ \}$
- $f^{M/E}([x]) = \bigcup_{y\in[x]} \{\ [a] \mid a\in f^M(y)\ \}$.

□

There is, of course, an implicit assumption about the form of the axioms in SP. In [102] SP is just a signature, but if axioms are also present, as in [53], then the theorem does not hold in full generality. For the deterministic models, it is shown in [74], that the model class of SP is closed under homomorphic images iff all the axioms of SP are positive formulae. It is easy to construct an example with inequality among the axioms which will show that M/E, for a given multimodel M, is not necessarily a multimodel. It is not therefore unreasonable to expect that the result from [74] will generalize to multimodels.

The importance of this nice counterpart of the analogous result for the deterministic equational classes is further diminished by the fact that one is still left with the problem of constructing a multimodel (the initial one?) from which one could start taking quotients.

Consistency relation

Equality interpreted as set equality means that two terms are equal if they *possibly can* return the same results. It does not, however, guarantee that they will. In terms of an arbitrary actual observation it does not guarantee anything. Following this line of thought of "the possible," one arrives at the notion of the equality as the sheer prospect that two terms may conceivably happen to return the same result.

[1] We do not focus on the distinction between visible and hidden sorts which, though formally important, would only add unecessary details to the presentation.

Example 2.16

Suppose we have a specification with two sorts V (visible) and H (hidden), and the operations binary choice $\sqcup$: H$\times$H$\to$H, and g: H$\to$V. Let A be a multistructure where

$H^A = \{\underline{a},\underline{b},\underline{c}\}$ $\quad V^A = \{\underline{0},\underline{1},\underline{2}\}$

$x \sqcup^A y = \{x,y\}$ $\quad g^A(\underline{a}) = \{\underline{0}\}$ $\quad g^A(\underline{b}) = \{\underline{1}\}$ $\quad g^A(\underline{c}) = \{\underline{2}\}$

We then have

$g(\sqcup.\{\underline{a},\underline{b}\})^A = \{\underline{0},\underline{1}\} \neq \{\underline{0},\underline{2}\} = g(\sqcup.\{\underline{a},\underline{c}\})^A$

If we stick to the strict model of observations then each observation involves only one unique result. Now, unless we impose some fairness conditions on $\sqcup$, it is free to display any degree of nondeterminism. We have no guarantee that the choice operator will produce all possible results. Since $\{0,1\}\cap\{0,2\}\neq\emptyset$, we cannot be sure that both $g(\sqcup.\{\underline{a},\underline{b}\})$ and $g(\sqcup.\{\underline{a},\underline{c}\})$ won't consistently produce the same result.

□

Thus, under this interpretation, $s=t$ if $\{s\}\cap\{t\}\neq\emptyset$. This relation is called *inseparability* in [53]. s and t are inseparable if both are capable of returning the same result, i.e., if there *is a possibility* that one may be unable to see the difference between them. This is not an equivalence relation. (It is not transitive, e.g., $\{s\}=\{0,1\}$, $\{t\}=\{1,2\}$, $\{p\}=\{2,3\}$.) The appropriate generalization requires the following definition:

E is a *consistent* relation iff

$$\forall (t,s)\in E,\ C[\_]\in W_{\Sigma,\{x\}},\ \exists a\in\{C[t]\}\ \exists b\in\{C[s]\} : (a,b)\in E \qquad (2.17)$$

Two terms are *consistent* [64, 53] if they belong to some consistent relation. (In particular, suppose that t and s are ground terms and $\{t\}$, $\{s\}$ are the sets interpreting t, s in a given structure A. If E on the individuals from A is equality and we let $[t]$ denote the class of terms equivalent to t with respect to the relation E, then $\bigcap\{\{s\} \mid s\in[t]\}$ must be nonempty. E.g., s, p, and t above cannot all belong to the same E-equivalence class because $\{s\}\cap\{p\}=\emptyset$.)

Consistent relations generalize the notion of congruence. A congruence relation is consistent and a consistent relation on a deterministic structure is a congruence. The same applies also to the extraction equivalencies. This is a frequently occurring phenomenon that different generalizations of deterministic concepts to the nondeterministic context all reduce to the same definition when restricted again to the deterministic situations.

Obviously, extraction equivalence implies consistency, and in fact, consistency is maximal among the considered interpretations of equality in the sense that it will identify every pair of terms which possibly (consistently) can be identified. Thus it corresponds to the maximal congruence for a deterministic structure and, in analogy with the latter, characterizes the terminal multimodels (see 4).

[53] contains more detailed and intricate notions of equivalencies and relations between them. We can also refer to the interesting work in [93, 95, 94] for a more "realistic" study of the nondeterministic equivalence where observations are defined relatively to a given programming language.

3. Initial models & specification languages

The success of the initial semantics for deterministic equational specification motivated many attempts at generalizing it to the situations where also nondeterministic operations are specified equationally. Below we review some of these attempts and arrive at the conclusion that purely equational specification of nondeterminism is insufficient for ensuring the existence of an initial model. Some more refined tools – at the syntactic as well as semantic level – are needed.

Consider an equational specification SP which is supposed to be given a power algebra semantics. As we have observed, power algebras are deterministic structures, and furthermore, any standard model of SP may be considered a power model by identifying all elements with 1-element sets and extending the operations pointwise. Thus, any equational specification will have a power model which is initial in the category (PMod,PHom). But what is missing in such a model is nondeterminism. The crucial thing for the semantics of nondeterminism is the distinction between the *individuals* and the *sets of individuals* which gets completely lost in PMod and PHom. In fact, an initial power model need not consist of 1-element sets. Any sets will do, since distinct sets are just distinct elements in PMod and PHom map sets to sets without considering their elements.

The semantic distinction may be introduced in various ways, for instance, by taking (PMod,MHom) or (PMod,EHom). In the former case, the same (deterministic) model as in (PMod,PHom) will be initial (provided MHom are tight), so things do not really get much more exciting. In the latter case, the deterministic power term structure will not be initial any more, since there will be several element homomorphisms from a model interpreting every term as a 1-element set to a model where some terms are interpreted as sets with more than one element.

What is needed is a closer relationship between the intuitive meaning of the power set structures (i.e., sets as the *result sets* of the operations) and the syntactic means of specifying the relations reflected in such structures.

3.1. Choice as a primitive – set union

A “minimalist” approach [63, 77] admits two primitives in the specification language: equality, which is interpreted as set equality, and binary choice which is specified by the join axioms:

J: $x \sqcup y = y \sqcup x$ (JC)
$x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$ (JA)
$x \sqcup x = x$ (JI)

Sometimes one also introduces a bottom element – the empty set [50] – and postulates distributivity of $\sqcup$ [63, 91]:

J1: $x \sqcup \perp = x$ (JE)
$f(..,x \sqcup y,..) = f(..,x,..) \sqcup f(..,y,..)$ (JD)

Notice that the last axiom ensures singular semantics. So far, we have not done anything new – $\sqcup$ is just a new deterministic operation specified equationally.

To make it into a *choice* one may attempt to require that $\sqcup$ is to be interpreted as set union. This, however, goes awry very quickly:

Example 3.1.a

The specification with three constants 0,1,2 and the axiom $0 \sqcup 1 = 0 \sqcup (1 \sqcup 2)$ has no initial model. For, constants must be interpreted as singletons $\{\underline{0}\},\{\underline{1}\},\{\underline{2}\}$,[1] and $\sqcup$ is set union, so $\mathcal{I}(0 \sqcup 1) = \{\underline{0},\underline{1}\}$ and $\mathcal{I}(0 \sqcup (1 \sqcup 2)) = \{\underline{0},\underline{1},\underline{2}\}$. In order to make these two sets equal we have to identify $\underline{2}$ with $\underline{0}$ or $\underline{1}$. Either choice leads to the situation where there is no homomorphism to the model obtained by realizing the other choice. Also, since choice is set union we cannot take $\mathcal{I}(0 \sqcup 1)$ to be the set $\{\underline{0},\underline{1},\underline{2}\}$.

□

[1] Actually, this depends on the definition of power set structure and homomorphism we are working with. PMod would give the possibility of interpreting them as arbitrary sets. They have to be different sets, though, and so the problem would persist anyway.

The problem arises from the fact that one has decided in advance what the meaning of $\sqcup$ should be. Since it *is* the set union there is no other way to make the sets $\mathcal{I}(0\sqcup 1)$ and $\mathcal{I}(0\sqcup(1\sqcup 2))$ equal than by identifying $\underline{2}$ with $\underline{1}$ or $\underline{0}$. Taking the quotient of the term structure or interpreting $0\sqcup 1$ as the set $\{\underline{0},\underline{1},\underline{2}\}$ are excluded. Thus axioms affecting the semantics of choice may introduce inconsistency (albeit not a logical one) with this pre-defined meaning.

The axioms J1 are given in [63] without requiring the interpretation of $\sqcup$ as set union. Then, of course, all equational specifications will have initial models, but this again is at the price of loosing control over the distinction between deterministic and nondeterministic operations. Nevertheless, conditions are given (in terms of the associated rewriting systems) under which initial models of such specifications are isomorphic to the expected multimodels with $\sqcup$ interpreted as set union. The conditions (left-linearity and $\sqcup$-freeness, see 7) amount to forbidding situations like the one from the example – $\sqcup$-freeness, for instance, disallows equating two terms which both contain occurrences of $\sqcup$.

The problem from example 3.1.a actually has two parts: One is that choice obtains some fixed semantics. Set union is perhaps not the best one and we will see in a moment other possibilities. The other is that the specification language does not provide sufficiently flexible means for introducing new, nondeterministic operations by reference to this primitive one. Equations are symmetric and affect equally both sides. Introduction of a new operation, f, defined by f=$0\sqcup 1$, and f=$0\sqcup(1\sqcup 2)$ would lead to the same problem. Thus each new equation is a potential source of a conflict.

It turns out that an increase in the expressive power of the specification language is required. Some refinement of the equational language is needed for ensuring that choice will be a nondeterministic operation if this is not secured by some predefined semantics. A more flexible language is needed to distinguish between the equivalence of two terms and the fact that one denotes only a (more deterministic) possible result of the other.

3.2. Inclusion

Since = is interpreted as set equality it shouldn't do much harm if we instead made use of set inclusion as the primitive relation between terms [59, 58, 127, 133, 130]. Hußmann supplies a very constructive generalization in [59, 58] where he introduces appropriate rewriting techniques. Inclusion is the only primitive operation in the language of [59, 58]. There specifications are sets of inclusion rules (we shall keep "$\subseteq$" as denoting set inclusion in the structures and use "$\prec$" for the corresponding syntactic relation):

Example 3.1.b

1) $0 \prec \sqcup.\{0,1\}$, $1 \prec \sqcup.\{0,1\}$

2) $0 \prec \sqcup.\{0,1,2\}$, $1 \prec \sqcup.\{0,1,2\}$, $2 \prec \sqcup.\{0,1,2\}$

Notice that choice is no longer a primitive operation. The problematic axiom now becomes:

3) $\sqcup.\{0,1\} \succ\!\prec \sqcup.\{0,1,2\}$ (writing $\succ\!\prec$ for two inclusions)

The initial model is given by the following interpretation

4) $\mathcal{I}(c) =\{\underline{c}\}$, for c=0,1,2

5) $\mathcal{I}(\sqcup.\{0,1\}) = \mathcal{I}(\sqcup.\{0,1,2\} = \{\underline{0},\underline{1},\underline{2}\}$

□

Since we have inclusions instead of equality in 1) the (initial) interpretation of $\sqcup.\{0,1\}$ can be expanded if necessary – here it contains $\underline{2}$ besides $\underline{1}$ and $\underline{2}$. Choice is no longer primitive - the semantics of sets is built into the semantics of $\prec$. In the initial (term) model, any term t is interpreted as the set of $[s]$ such that $s\prec t$, where $[s]$ is the equivalence class of s, i.e., the set of all p such that $s\succ\ \prec p$.

However, to ensure the existence of the initial models some assumptions about the specification must be made [59, 58]. One is that the specification has to possess a deterministic basis, i.e., a set of terms which are interpreted as singletons, and such that every term can be reduced to at least one deterministic term (DET-completeness). To express this a predicate DET is introduced into the language, where DET(t) is valid if the term t is deterministic. Another assumption (DET-additivity) implies that arguments in the right hand sides of inclusions must be deterministic. For instance, instead of $0 \prec f(1 \sqcup 2)$ we must write $0 \prec f(1)$ and $0 \prec f(2)$. Under these assumptions, every specification has a (loosely) initial model in (MMod,MHom) which can be constructed as in the above example. (These conditions are further discussed in section 7.)

3.3. Partial orders

Instead of sticking to the set interpretation one may generalize the fact that the set union (choice) is join operator. That is, instead of set union we can speak about join, instead of inclusion – about a partial order.

The first work exploring these ideas in an algebraic framework is [77]. Given the axioms J (from 3.1) and a language with = and $\sqcup$, one can define a structure on the models reflecting the intended nondeterministic interpretation. For each sort in the signature Σ, the *result partial* order on the word structure W_Σ, $\prec$, is defined [77, 49]:

R: $s \prec t$ iff
1. $s \doteq t$ or
2. $t \doteq s \sqcup s_1$ or
3. $s \doteq f(p_i)$, $t \doteq \mathrm{f}(r_i)$, and $p_i \prec r_i$

This definition amounts to the standard definition of a partial order on a lattice extended with the monotonicity axiom 3. which ensures that an increase in nondeterminism of the arguments never decreases the nondeterminism of the result. Note that the singularity axiom JD does not follow from J+R. We only have that R3 implies $f(\ldots,x,\ldots) \sqcup f(\ldots,y,\ldots) \prec f(\ldots,x \sqcup y,\ldots)$, reflecting our intuition that whatever can be produced under singular semantics can also be produced under the plural one.

Equational specifications with $\sqcup$ (and the axioms J) are just classical equational specifications and hence always possess initial models. These models are essentially deterministic and nondeterminism is *encoded* as the additional structure $\prec$ implied by the occurrences of $\sqcup$. However, $\prec$ is not a part of the specification language.

Example 3.2.a

Let us suppose that we want to add a new operation f to the specification (J) with the following axioms:

1) $f(x_1,x_2,x_3) = x_1 \sqcup x_2 \sqcup x_3$
2) $f(0,1,2) = f(0,0,1)$

We will obtain an initial model where $0 \sqcup 1 = 0 \sqcup 1 \sqcup 2$. This does not cause the problems of example 3.1.a since we now have a deterministic structure and merely take quotients of such structures. However, as a result we will obtain that $2 \prec 0 \sqcup 1$ holds, which is perhaps not quite what we had in mind.

□

It may be slightly surprising that defining a *new* operation has such consequences for the original structure. After all, writing 1) and 2) we might be interested not in changing the semantics of choice but only in defining a new nondeterministic operation which behaves a little differently in some cases - for instance, so that it never returns some “forbidden” values. This illustrates the failure of this “deterministic” approach to reflect, in the general case, the intended meaning of nondeterministic choice as set union. The specification from example

3.2 does not conform to the restrictions from [63] mentioned at the end of 3.1 which make models of specifications with = and $\sqcup$ isomorphic to the intended models where $\sqcup$ is interpreted as set union.

The collapse of the original ordering results here from the exclusive use of equations. The approach from 3.2 can easily accommodate such a situation. Using inclusions instead of equality and making $\sqcup$ a defined operation allows one to specify the nondeterministic operations directly.

Example 3.2.b

The corresponding specification would be:

1) $x_i \prec f(x_1,x_2,x_3)$, for i=1,2,3

2) $f(0,1,2) \succ\!\prec f(0,0,1)$

The initial interpretation will assign the set $\{\underline{0},\underline{1},\underline{2}\}$ to both $f(0,1,2)$ and $f(0,0,1)$. This does not affect semantics of other operations, like $\sqcup$, which are specified independently.

□

The resulting partial order $\prec$ is introduced in [77] in addition to the continuous structure on the models needed for finding the solutions of recursive systems of equations. We devote section 5 to this topic and will continue the discussion of the present work there. At this point we may only remark that as the consequence of using continuous algebras the model class is no longer closed under substructures and quotients. Thus initial models can not be constructed simply as quotients of the word structure.

3.4. Lattices and unified algebras

The approach discussed above introduced axioms for the join operation and defined (implicitly) the corresponding partial order. Example 3.2.a-b indicated that it might be advantageous to reflect the semantically defined partial order in the specification language. This leads us directly to the lattice structures. In a very general form, they have been proposed as the theoretical foundation of data types in [115, 116]. In [6] lattices are used to define a specification language enabling the construction of the weakest predicate transformers for commands which include angelic and demonic nondeterminism, and in [1] a relational approach to data types uses lattices as the basis of the specification language and relational calculus. Here we will consider a very elegant construction of *unified algebras* introduced by Mosses in [92, 91]. Unified algebras combine the advantages of several approaches described so far, adding new interesting features.

Every *unified signature* Σ contains the subsignature Ω with the operations {nothing, _|_, _&_} and predicates {_=_, _≤_, _:_}. For the sake of notational compatibility with the rest of this exposition we will use the symbols $\{\bot, \_\sqcup\_, \_\sqcap\_\}$ and $\{\_=\_, \_\prec\_, \_:\_\}$. Formulae of the specifications are (universal) Horn clauses. A *unified Σ-algebra A* is a structure (with one sort) such that:

- $|A|$ is a distributive lattice with $\sqcup^A$ as join, $\sqcap^A$ as meet, and $\bot^A$ as bottom.
- There is a distinguished set $E^A \subseteq |A|$ – the *individuals* of A.
- $=^A$ is the identity on the elements of the lattice (not only on the individuals).
- $\prec^A$ is the partial order of the lattice, i.e., $x \prec^A y$ iff $x \sqcup^A y =^A y$.
- For every $f \in \Sigma$, f^A is monotonic wrt. $\prec$.
- $x :^A y$ holds iff $x \in E^A$ and $x \prec^A y$.

“Unified” refers to the fact that there is only one syntactic sort, and no syntactic distinction is made between sorts and individuals. Sorts in the sense of classical algebra, as well as individuals, are elements of the lattice. Also sorts and nondeterminism are treated in a unified way. The partial order of the lattice corresponds to the set inclusion; sorts and nondeterministic choice are interpreted as joins – the elements “just above” their members. The set E represents the individuals. The individuals need not be the atoms of the lattice, i.e., the elements “just above” the bottom, though, typically, this will be the case. _:_ is the membership of singletons – t:s means that t is contained in s ($t \prec s$) and t is an individual. Monotonicity with respect to the partial order of the lattice allows the extension of the definitions of the operations on individuals to their respective upper bounds (sorts or nondeterministic choices).

The revision of example 3.2 will look as follows:

Example 3.2.c

The richer semantics gives us several possibilities to interpret the problem.

1) $x_i \prec f(x_1,x_2,x_3)$, for i=1,2,3

2) $f(0,1,2) = f(0,0,1)$,

This specification will lead to the initial model where 2 is a possible result of (lies “below”) $f(0,0,1)$. The alternative

1′) $f(x_1,x_2,x_3) \prec \sqcup.\{x_1,x_2,x_3\}$

2′) $f(0,1,2) = \sqcup.\{0,1\}$

will ensure that the result set of $f(0,1,2)$ is the join of 0 and 1 and does not include 2. All the other applications of f will be “just below” the corresponding $\sqcup$ but their only result in the initial model will be the bottom element.

□

Axioms 1′-2′ yield a specification which, in the terminology from 3.2, is not DET-complete, and hence does not have an initial multimodel. This fact is reflected in the initial unified algebra by the $\perp$ element which corresponds to an unspecified result set of $f(x_1,x_2,x_3)$.

Of course, it can still happen that the ordering collapses as it did in example 3.2.a (e.g., take 1) and 2′). Although choice is a primitive operation of unified algebras, the specification language gives one the ability to avoid (or introduce) such cases at will.

The “inclusion” approach from 3.2 can be subsumed under unified algebras. Thus DET(t) corresponds to t: t and inclusions to the partial order, and the other way around, t: s is the same as $t \prec s \wedge$ DET(t).) The initial unified algebra model of 1)-3) from example 3.1.b will essentially be the same as the one presented there (a lattice with individuals $\underline{0}$, $\underline{1}$ and $\underline{2}$, and the top element $\sqcup.\{\underline{0},\underline{1}\}=\sqcup.\{\underline{0},\underline{1},\underline{2}\}$). In fact, both will give the same class of models when interpreted in multialgebras. The analogy can be further illustrated by the fact that both approaches emphasize the distinction between individuals and sets and, in fact, treat the distinction in quite similar ways.

Example 3.3

	Unified algebra	Hußmann’s multialgebra	
	0:0, 1:1,	DET(0), DET(1),	$0 \prec 0\sqcup 1$, $1 \prec 0\sqcup 1$
1.	c: $0\sqcup 1$	DET(c), $c \prec 0\sqcup 1$	
2.	d=$0\sqcup 1$	$d \succ\prec 0\sqcup 1$	

In 1. c is a nondeterministic constant. In the initial model it is an (additional) element below $0\sqcup 1$, and in any model where 0 and 1 are the only individuals it will be equal to either of them (to which, will vary from model to model). d in 2. is a nondeterministic operation equivalent to $0\sqcup 1$, i.e., it is the same element in the lattice as $0\sqcup 1$.

□

The distinction between individuals and sets cannot be expressed if equality is the only primitive of a language. But it should be obvious that it may be essential to know whether we are speaking about the result returned by a particular application of a nondeterministic operation, or about the *set* of such results.

In the last example we show a feature of the unified algebras which brings us closer to the explicit treatment of the singular vs. plural semantics. The basis for the distinction is the possibility of defining functions either by the pointwise extension from their definition on individuals or by direct definitions on the non-individual arguments.

Example 3.4

Define *if_then_else_*:

i1) if $\mathbb{T}$ then x else y = x, if $\mathbb{F}$ then x else y = y,

i2) if $z \sqcup w$ then x else y = (if z then x else y) $\sqcup$ (if w then x else y)

and

3) $f(x)$ = if x=x then 0 else 1.

Then for all values of x, $f(x)=0$. In particular $f(0\sqcup 1)=0$ since the equality x=x holds for all elements of the lattice. This is the result required by the singular semantics of the argument in $f(x)$.

Let us define a new equality predicate $\approx$:

e1) $x{:}x \Rightarrow (x\approx x) = \mathbb{T}$ e3) $(x\approx y) \prec \mathbb{T}\sqcup\mathbb{F}$

e2) $(0\approx 1) = \mathbb{F}$

Monotonicity of the operations and axiom e3) give then

$(0\sqcup 1\approx 0) = \mathbb{T}\sqcup\mathbb{F}$ and $(0\sqcup 1\approx 1) = \mathbb{T}\sqcup\mathbb{F}$

and hence

$(0\sqcup 1\approx 0\sqcup 1) = \mathbb{T}\sqcup\mathbb{F}$.

Observe that the antecedent in e1) forces $x\approx x$ to be true only when x is an individual.

Now change 3) to

3′) $f'(x)$ = if $(x\approx x)$ then 0 else 1

Since $(0\sqcup 1\approx 0\sqcup 1) = \mathbb{T}\sqcup\mathbb{F}$, i2) implies that $f'(0\sqcup 1) = 0\sqcup 1$ reflecting the plural meaning of the argument in $f'(x)$.

□

This example illustrates the power of unified algebras in the treatment of nondeterminism which is unmatched by any other approach we have discussed. On the one hand this feature is related to the ability to distinguish between individuals and sets – but now with respect to variables. In [58, 59] the distinction was applicable only to terms which were not single variables. For a given non-variable term (say, $f(x)$) one could write DET($f(x)$) (to make $f(x)$ deterministic) or not (to allow $f(x)$ to include several results), but for variables the axiom DET(x) was always a given. On the other hand, this is a consequence of the fact that the variables in unified algebras refer to arbitrary elements of the lattice, and these elements may represent individuals as well as sets. Consequently, operations are in general interpreted as in power models, i.e., map sets to sets. Singular semantics is obtained as indicated in 2.1 by taking the pointwise extension of the operations defined on the individuals.

The main advantage of unified algebras is that specifications using at most Horn clauses always have initial models (and the operations from Ω can be specified by Horn formulae). Other advantages follow from the general properties of institutions [41, 112, 122] in which unified algebras are defined. In particular, the institution of unified algebras is liberal so that one can impose (under a slightly modified definition of the forgetful functor) various data constraints, such as the fact that $\mathbb{T}\neq\mathbb{F}$.

3.5. The price of initiality

We have seen that multialgebraic semantics does not, in general, admit initial models and that one has to introduce some partial order (lattice) structures to guarantee the existence of such models. We end this section by observing some disadvantages of the initial semantics in modeling nondeterminism.

Partial orders provide simpler mathematical structures than multialgebras. However, this is a consequence of the fact that the distinction between individuals and sets – which is the underlying intuition of the distinction between deterministic and nondeterministic operations – gets blurred. Consequently, the partial order models sometimes fail to correspond to any set based structure which would express the intended meaning of a nondeterministic specification.

Taking choice as the primitive (join) may lead to the unintended collapse of the ordering if one is not sufficiently careful in writing the axioms. Furthermore, the minimal elements of a partial order need not correspond to the actual individuals. A specification with two nondeterministic constants a and b, and the axiom $a \prec b$ does not have an initial multimodel because the result set of a (and b) may be arbitrary. The initial partial order model will have just one (minimal) element $\underline{a}$ which is below $\underline{b}$. The price of this generality is that $\underline{a}$ and $\underline{b}$ are not sets and, in particular, $\underline{a}$ is not an element of the set $\underline{b}$.

More significantly, and this applies to multialgebras as well as partial orders, initial models lead to an appearance of additional elements not intended by the specifier [128]. In example 3.3 the axiom c: $0 \sqcup 1$ (or DET(c), $c \prec 0 \sqcup 1$) was given for the constant c. Its intended meaning is probably that c is an operation which is either 0 or 1, but we do not (want to) state which. Because of this "do not know which", the initial model has to introduce, in addition to the elements $\underline{0}$ and $\underline{1}$ the extra element $\underline{c}$, and all three elements appear as the possible results of $0 \sqcup 1$. Although formally everything is correct as far as initiality is concerned, we may feel justified in calling elements such as $\underline{c}$ "intuitive junk" and thus a violation of the "no junk" dictum which favors initial models in the deterministic case.

The specification of c attempts to indicate that c is equal to 0 or 1 without actually identifying c with any of the two. One wants to model the fact that "c=0 *or* c=1" without spelling it clearly out. This fear originates, of course, in the knowledge that disjunctive equations do not, in general, allow initial models – Horn-formulae are the most general ones admitting initial models [78, 76, 122].

The source of the problem is the presence of *underspecification* and that one insists on using initial models of such specifications, not with nondeterminism per se – deterministic underspecifications give rise to similar problems. However, in the deterministic case underspecification is an optional tool for handling exceptional situations rather than the standard procedure. Typically, one tries to apply some strategy, such as generator induction [28, 44, 60], resulting in specifications with intended initial models. Nondeterminism, on the other hand, implies disjunction and, with it, underspecification.

Another aspect of this implication is that the nondeterministic operations themselves are not adequately specified. In all three cases, underspecification of the c element destroyed $0 \sqcup 1$ as binary choice and turned it into an operation capable of returning, besides 0 and 1, some c. These observations make us argue that making a clear distinction between nondeterminism and underspecification and admitting disjunctive formulae to handle the latter is more advantageous than it is dangerous. Above all, it would allow us to restore the intended simplicity of the model.

Introducing the lattice structure in unified algebras one avoids disjunction and models it with the help of the join operation. It might seem that, since disjunction is the join operation (in the Boolean algebra of propositions), the two will lead to similar results. However, the obvious differences are quite significant:

Lattice/Joins	Disjunctive specification
all joins must be present	only relevant disjunctions are included
introduces "junk"	eliminates "junk"
allows initial models	disallows initial models

The first point refers to the fact that in the lattice model join is an operation (choice) with predefined semantics. If there are several constants in the specification, say d besides 0, 1 and c, then unified algebra is forced to interpret the element $a\sqcup d$ as join. But if we are interested in a binary operation which is only some variation of choice, e.g., chooses nondeterministically merely between 0 and 1, and for other arguments behaves as, say first projection, then we have to relate it to the primitive choice operation. $0\sqcup d$ will always be there. Disjunction allows us to specify different kinds of nondeterministic (also partially defined) operations without any reference to some basic form of nondeterminism. The significance of disjunctive specifications has been noted by several authors. It has been studied in the theory of data bases [37, 124, 125], introduced in the theory of algebraic specifications and conditional term rewriting [42, 137, 62], and utilized for specifications of nondeterminism [127, 128, 126].

We may treat disjunction as a replacement of the join operation. It will not lead to the lattice models, but instead to the standard algebraic models. The most important advantage of this is that it actually removes the unintended "junk" and produces models which have an intuitive clarity comparable to initial models of deterministic specifications. We would (under)specify our example in the obvious way:

Example 3.5

1) Det(0)   Det(1)   Det(c)
2) $x\prec x\sqcup y$ $y\prec x\sqcup y$
3) $c=0 \vee c=1$
4) $d \prec a\sqcup b$

□

Notice that the choice operator is actually not needed at all in the specification of c. Disjunction allows us to distinguish c as underspecified from d as a result returned by nondeterministic choice (although we are not going into the details here – see [127]).

The specification of example 3.5 does not have an initial model. Nevertheless, disjunction does not spoil initiality completely. Disjunctive specifications always possess *quasi-initial* computational semantics, and mild restrictions on the specifications guarantee existence of quasi-initial multialgebra semantics [127]. Quasi-initiality generalizes initiality to situations involving "either ... or ...". For the above example, such a semantics can be summarized by saying that the model class will consist of two parts: the models where c equals 0 ($M0$) and those where it equals 1 ($M1$). Two models, A and B given by

$$\begin{tikzcd}
\mathrm{A} & & \mathrm{B} \\
\underline{0} & a \arrow[l] \arrow[r] & \underline{0} \\
\underline{1} & b \arrow[l] \arrow[r] & \underline{1} \\
\{\underline{0},\underline{1}\} & a\sqcup b \arrow[l] \arrow[r] & \{\underline{0},\underline{1}\} \\
\underline{0} & c \arrow[l] \arrow[r] & \underline{1}
\end{tikzcd}$$

will be initial in the respective components $M0$ and $M1$ of the model class.

Now, the difference between an underspecified c and a nondeterministic operation $\sqcup$ is that the former has a unique value in every model, but these values may vary from one model to another. The latter, on the other hand, may also return different values in one model. $0\sqcup 1$ may be thought of as a series of distinct applications, each returning either 0 or 1. But this means that every single application of $0\sqcup 1$ is underspecified! The final example makes use of this fact to specify $\sqcup$ as a user-defined operator.

Example 3.6

1) Det(0)   Det(1)   Det(c)   Det(d)
2) $x=y\sqcup z \;\Rightarrow\; x=y \vee x=z$
3) $c=0 \vee c=1$
4) $d \prec 0\sqcup 1$

□

Axiom 2 specifies binary choice by underspecifying each particular application of it. It reads: if x is a result of (some) application of $y\sqcup z$ then x equals y or z. (Compare this to the axiom 2. from example 3.5.) Substituting 0, 1, d for y, z, x, and using axiom 2. we will get that d equals 0 or 1. The initial covering of the model class will contain the following models: Each of the models A and B above will now be split into two new models depending on whether $\mathcal{I}(d)=\underline{0}$ or $\mathcal{I}(d)=\underline{1}$. In addition, there will be models where $\mathcal{I}(0\sqcup 1) = \{\underline{0}\}$, and $\mathcal{I}(0\sqcup 1) = \{\underline{1}\}$ (these can be excluded by adding axioms 2. from example 3.5). Since d is defined in terms of $0\sqcup 1$, in models where one of these hold d will be equal to the respective $\mathcal{I}(0\sqcup 1)$. c remains underspecified and so even if $\mathcal{I}(0\sqcup 1)=\{\underline{0}\}$ c may still equal 1. We believe that all these models reflect the intuitive possibilities of the specification. In particular, none will contain any redundant elements, and the sort will consist of at most $\underline{0}$ and $\underline{1}$.

The price for this adequacy is, of course, that we no longer have initial semantics. This is the trade-off between using disjunction in specifications or excluding it.

The argument favoring disjunctive equations is bolstered by the following observation: The two axioms $0\prec 0\sqcup 1$ and $1\prec 0\sqcup 1$ may be given initial semantics where 0 and 1 are the only results of $0\sqcup 1$. However, they only say that 0 and 1 *must* be among the possible results, not that they are *the only* ones. In the non-initial models of the class any kind of elements may occur as the additional results returned by $0\sqcup 1$. Thus, simple (or Horn) formulae can only specify "lower bounds" of nondeterminism which coincide with the "upper bounds" only in the initial models.

Horn-specifications also lead to "necessarily" nondeterministic implementations. The axioms $0\prec t$ and $1\prec t$ force *all* models to be nondeterministic (unless 0 is identified with 1). They would disallow a model which implements t as a deterministic operation returning always only 0 (or only 1). Implementations of a given algebra may restrict only nondeterminism which is not explicitly specified by the axioms. Thus one is forced to distinguish between nondeterminism and underspecification, and can not use nondeterminism as a pure abstraction mechanism at the specification level.

4. Terminal models

The initial semantics has several well-known limitations. 1) As remarked in the last section, in order to ensure the existence of initial models one has to restrict the specification language to conditional equations. 2) On the other hand, initial models yield only special classes of algebras (semicomputable, or, in the case of monomorphic specifications, computable ones, see [12, 14]). 3) It has also been pointed out [56, 19, 21] that, typically, initial structures are not *fully abstract* – they often distinguish elements which behave identically in all contexts, and additional axioms are needed if one wants to identify them.

Terminal algebra semantics shares two of these limitations. 1) Not all specifications possess terminal models. According to [17], the restriction of the specification language to the positive formula (universally or existentially quantified disjunctions or conjunctions of positive atomic formula (equations)) guarantees the existence of terminal models. This may easily reduce to the trivial model but the result admits positive specifications relative to some primitive (monomorphic) types as long as the specifications are sufficiently complete wrt. these primitive types. Thus, required inequalities may be introduced via the primitive types.

2) Furthermore, the data types obtained by terminal semantics are co-semicomputable [12, 14].

An advantage of the terminal models is that they do not distinguish elements which are observationally equivalent. Every other model M can be seen as a refinement – an implementation – of the terminal model Z, and the unique homomorphism $M \rightarrow Z$ serves as the natural abstraction function which determines the equivalence on the concrete representations of the abstract values. Focusing on observational equivalence, terminal models are often well suited for studying implementation relations.

These properties are carried over to the terminal models of nondeterminate specifications. While constructing initial power set models of nondeterminism poses severe problems, one has at least obtained a characterization of the congruence determining the terminal multimodel. Let B be a multimodel of a specification SP, and $L(B)$ be the class of multimodels of SP such that for every $M \in L(B)$ there exists a (loose) multihomomorphism $M \rightarrow B$.

Theorem 4.1 [53]. A multimodel B is terminal in the class ($L(B)$,MHom) iff it identifies all consistent values.
□

Thus, recalling the definition of consistency from 2.17, a terminal model has to identify two values whenever there is a *possibility* that they may give rise to the same observations in all contexts.

The theorem is qualified by the clause "B is terminal in $L(B)$" which means that we still do not have the criteria for the existence of terminal models. In particular, the work in [53] has a purely semantic character and no formal specification language is considered. We may, however, use the theorem to show that a terminal model does not exist for a specification SP by showing that the class of multimodels of SP, MMod(SP), is not equal to the class $L(B)$ for any model B which identifies all consistent values.

In [21] terminal semantics is used to study the implementation relation between four kinds of nondeterminism. The programming languages CN, BN, AN, and LN are specified which include, respectively, erratic, demonic, angelic, and loose nondeterminism (which, roughly, reflects the idea of a specification with loose deterministic semantics). For the definition of these notions a predicate "*loops*" is introduced which expresses the possibility of nontermination ($loops(s)=\mathbb{F}$ iff s always terminates). The terms correspond to the statements of the intended programming language, and the models of the specifications represent the possible semantics of the languages. It is shown that the first three, CN, AN, and BN, have terminal models. These are of special interest because they are

1. *minimally defined*, i.e., whenever there is a semantics in which some statement s is undefined, then this possibility is reflected in the terminal model where s is undefined too, and
2. *fully abstract*, i.e., two statements which have the same *loops* values and the same result sets are equal.

LN does not have a terminal model, but all minimal models of LN are possible *deterministic* semantics of LN and are possible implementations of some of the terminal models of CN, AN or BN. The implementation ordering $\leq_I$ on models is defined in the natural way [93, 53, 18]:

$M \leq_I N$ iff for all terms t, elements a:

$$loops(t^N)=\mathbb{F} \quad \Rightarrow \quad loops(t^M)=\mathbb{F}$$
$$a^M \in \{t^M\} \quad \Rightarrow \quad a^N \in \{t^N\}$$

That is, M implements N if it terminates at least in the situations when N terminates and is not more nondeterministic than N. The result states then that the terminal models for AN and BN are incomparable but both are possible implementations of the terminal model of

CN. This reflects exactly our intuition about the relation between these forms of nondeterminism. Erratic nondeterminism (CN) is the most general and both angelic (AN) and demonic (BN) ones are possible specializations. Angelic and demonic nondeterminisms are, of course, mutually exclusive – neither subsumes the other. Finally, any nondeterminate operation which does not require fairness may be implemented (loosely) by a deterministic operation.

We may look at this result as a semantic counterpart of the complexity analysis of nondeterministic computations from [25]. Different kinds of nondeterminism are there classified according to the complexity of their *loops* predicate. Unbounded erratic nondeterminism is the most complex class which can be fully characterized only by Σ^1_1 formulae, while angelic nondeterminism (bounded as well as unbounded) requires only Π^0_1 formulae. Of course, the deterministic programs are simplest in this respect – they do not require the *loops* predicate at all since it can be uniquely determined from their input-output relation (which in all cases is of the same, partial recursive complexity, Σ^0_1). Other, more specific forms of biased nondeterministic computations are also included in the classification in [25].

The reader is referred also to [19, 20] for more details about the general approach on which the results from [21] are based.

5. Solutions of recursive equations – continuous models.

One of the reasons for the great importance of the initial models is that they, in a sense, “represent” the whole class of models. In particular, equality on the ground terms in an initial model “represents” the equality on such terms in all models. In order to check whether a ground equation holds in some model A we can check whether it holds in the initial one, and if it does conclude that it holds in A. Equations of a specification define the operations which are legal interpretations of the operation symbols. In particular, recursive equations define operations – namely operations which, when substituted for the operation symbols in the equations, provide a solution to the equational system. The next natural question is: can such solutions be constructed effectively?

The theory of the deterministic languages and specifications knows the “Mezei-Wright-like results” (after [88]) of the form: there is a (symbolic, initial) structure W such that:

1. solving a system of recursive equations over W and interpreting the solution in another structure A, and
2. interpreting the system in A and solving it directly

give the same result. One of the main results of this kind for the equational specifications is given in [40] for W the initial continuous model.

We will summarize briefly the ideas of such a construction and some difficulties in extending it to the nondeterministic case. The issue is especially relevant because it focuses on the problems of nontermination, finite observability and computability and allows us to identify some connections between the algebraic and the denotational approaches to semantics.

5.1. Deterministic preliminaries

Let Σ be a determinate signature. Any term $t \in W_{\Sigma,X}$ can be considered a tree $\hat{t}$ reflecting its syntactic structure. A term defined by recursion may be considered as a, potentially infinite, tree obtained by the stepwise unfolding of the term along its defining equations. An additional symbol $\perp$ is used to indicate the “unfinished” unfolding.

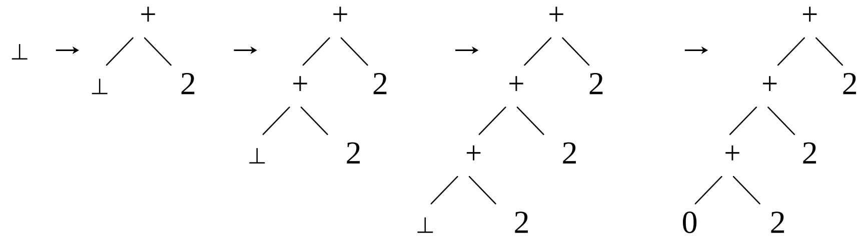
Example 5.1

Define

$f(0) = 0 \qquad f(x+1) = f(x)+2$

The tree for $f(3)$ is constructed by applying the definition to $\perp$ until no $\perp$ elements occur in the tree:

$$f(3) \rightarrow f(2)+2 \rightarrow (f(1)+2)+2 \rightarrow ((f(0)+2)+2)+2 \rightarrow ((0+2)+2+2)$$



□

If there is a possibility of infinite unfolding, then the semantics is obtained as the fixed point of the defining equations. For this purpose, the structure of Σ-algebras is enriched with the appropriate ordering to make them complete partial orders (cpo).

An infinite unfolding may lead to an *infinite term* so the carriers must contain elements interpreting such terms. The ordering must ensure that unfoldings constitute increasing chains: in particular, $\perp$ is the least element, and the operations are required to be continuous. Then, by the Knaster-Tarski theorem (e.g., [75]), unfoldings have the least fixed points which may be computed starting with the least element $\perp$.

Such a *computational* (*approximation*) *ordering*, $\sqsubseteq$, on terms $W_{\Sigma,\perp}$ is defined as follows [40, 77, 23, 96, 15, 97]:

C: $s \sqsubseteq t$ iff
1. $s \doteq t$ or
2. $s \doteq \perp$ or
3. $s \doteq f(p_i)$, $t \doteq f(r_i)$, and $p_i \sqsubseteq r_i$

Notice the natural interpretation of the unfolding in the above example as a gradual approximation to the final result by successive evaluation of the "currently undefined" $\perp$ leaves. This intuition is reflected by $\sqsubseteq$ in that a term s is $\sqsubseteq$-less (computed) than t if t can be obtained from s by substituting "better defined" terms for some occurrences of $\perp$ in s.

Every model $A_\perp \in Alg_{\Sigma,\perp}$ is required to be $\sqsubseteq$-continuous (i.e., the carrier is a $\sqsubseteq$-cpo and all operations are $\sqsubseteq$-continuous). Similarly, the homomorphisms of models are strict $\sqsubseteq$-continuous Σ-homomorphisms. The point of these requirements is that when all operations are $\sqsubseteq$-continuous the unfoldings are chains and have lub, and such lubs are preserved by the homomorphisms between models.

Technically, $\sqsubseteq$-continuous models are obtained by *ideal completion* [75, 40] of standard models partially ordered by $\sqsubseteq$. Every partially ordered algebra $A_{\Sigma,\perp}$ with a least element $\perp$ (for each sort) can be embedded into a continuous algebra $\hat{A}_{\Sigma,\perp}$ with the carrier being the set of all ideals of $A_{\Sigma,\perp}$ (i.e., downward closed subsets J such that any two elements of J have an upper bound in J) ordered by set inclusion. The embedding is the mapping $\hat{}: A_{\Sigma,\perp} \rightarrow \hat{A}_{\Sigma,\perp}$ which sends every element a of $A_{\Sigma,\perp}$ to the ideal generated by a, i.e., the set of all $b \in A_{\Sigma,\perp}$ such that $b \sqsubseteq a$.

Intuitively, we may think of this construction as adding the "infinite elements" as the limit points of chains in $A_{\Sigma,\perp}$. In particular, for the initial word structure $W_{\Sigma,\perp}$ we obtain the structure $\hat{W}_{\Sigma,\perp}$ (of finite and infinite terms) which is initial in $Alg_{\Sigma,\perp}$. Consequently, the least solution of recursive equations can be constructed as 1) the lubs of the unfolding chain in $\hat{W}_{\Sigma,\perp}$ and mapped by the unique homomorphism to some actual algebra $\mathrm{A}_\perp$, or else 2) the terms from the equations can be interpreted in the algebra and the solution can be found as a the lub of the chain in $\mathrm{A}_\perp$. The content of the Mezei-Wright-like theorem can be summarized as:

Theorem 5.2 [40]. Given a system of recursive equations over $W_{\Sigma,\perp}$, the following diagram commutes for every $A_\perp \in Alg_{\Sigma,\perp}$:

$$\begin{tikzcd} & W_{\Sigma,\perp} \arrow[dl, "\hat{}"'] \arrow[dr, "2."] & \\ \hat{W}_{\Sigma,\perp} \arrow[rr, "1."] & & A_\perp \end{tikzcd}$$

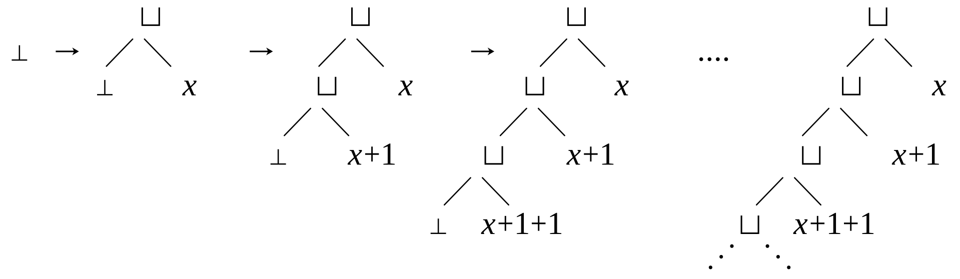
□

5.2. CPOs for nondeterministic choice

Let $(\Sigma,\sqcup,\perp)$ denote a (determinate) signature Σ extended, as above, with $\perp$, and with the nondeterministic binary choice $\sqcup$. The first generalization of 5.2 to nondeterminism is given in [77] which we have discussed briefly in 3.3. The generalization is quite straightforward. The ordering $\sqsubseteq$ is defined by the same axioms C and so lubs exist and are computed in the same way regardless of whether terms involve occurrences of $\sqcup$ or not.

Example 5.3

Define $f(x) = f(x+1) \sqcup x$. The (infinite) lub tree for $f(x)$ is the fixed point of the same unfolding process as in example 5.1.



□

Nondeterminism is encoded in the result ordering $\prec$ which is defined by the axioms R from subsection 3.3. Unlike in the (Hoare, Smyth, Plotkin) power domain constructions, the two orderings are not merged into one but are kept apart. Turning $\prec$ into a cpo might present new difficulties, such as an additional bottom element, and would bring us very close to the denotational definitions of the combined orderings. Instead, it is shown that the ordering $\prec$ is "compatible" with $\sqsubseteq$ in the sense that, if $\{a_i\}$, $\{b_i\}$ are two $\sqsubseteq$-chains such that $a_i \prec b_i$, for all i, then $a \sqsubseteq b$, where a, b are the $\sqsubseteq$-lubs of the respective chains. The intuitive content of this "compatibility" is that the two computations (computational chains) which at each step have $\prec$-ordered results are $\sqsubseteq$-ordered as computations.

Recall that axiom R3 (from section 3.3) made all the operations monotonic with respect to the result ordering. Homomorphisms are now required to be not only strict and $\sqsubseteq$-continuous but also $\prec$-monotonic. In general, quotients of continuous algebras are not continuous. However, it may be shown that in the special case of the congruence generated by the axioms JC, JA, JI (section 3.1) the quotient $\hat{T}_{\Sigma,\sqcup,\perp}$ of $\hat{W}_{\Sigma,\sqcup,\perp}$ is initial in the class of $\sqsubseteq$-continuous, $\prec$-monotonic models. This initiality result yields the desired analog of theorem 5.2.

The simplicity of this generalization relies on the fact that models are here, as we said, essentially deterministic. Also, the nature of $\sqcup$, although modeled to some extent by the result partial order, does not fully correspond to choice. There will be cases (and not very rare, as a matter of fact, for instance, the one from example 3.1) when a model of $\sqcup$ does not correspond to the intuition of choice. This means, that the obtained result does not have the desired consequences if we attempt to interpret specifications in some power set structures.

This may be explained by the comparison of the intuitive meaning of the tree $\hat{t}$ when the term t is determinate and when it contains applications of $\sqcup$.

Example 5.4.a

Let b: $\to$ S, f: S$\to$S, and g: S$\times$S$\to$S be determinate operations, and consider the trees for the terms

$t \doteq f(g(\bot,b))$: and $s \doteq f(\bot \sqcup b)$:

$$\mathrm{t} \doteq \begin{tikzcd}[column sep=small] & f \arrow[d, dash] & \\ & g \arrow[dl, dash] \arrow[dr, dash] & \\ \bot & & b \end{tikzcd} \qquad \hat{\mathrm{s}} \doteq \begin{tikzcd}[column sep=small] & f \arrow[d, dash] & \\ & \sqcup \arrow[dl, dash] \arrow[dr, dash] & \\ \bot & & b \end{tikzcd}$$

$\hat{\mathrm{t}}$ represents an incomplete evaluation which requires a substitution of some closed term for $\bot$ in order to represent the unique result of the computation (singular semantics is assumed). The intuitive interpretation of $\hat{\mathrm{s}}$, on the other hand, would say that the $\sqcup$ node represents a branching of the computation which may proceed either toward b – in which case we have a complete computation, or toward $\bot$ – in which case we do not have a one. It would not consider s as a single syntactic tree $\hat{\mathrm{s}}$ but rather as a pair of trees, each representing a possible result of the evaluation of s:

$$\hat{\mathrm{s}}_1 \doteq \begin{tikzcd} f \arrow[d, dash] \\ \bot \end{tikzcd} \qquad \hat{\mathrm{s}}_2 \doteq \begin{tikzcd} f \arrow[d, dash] \\ b \end{tikzcd}$$

□

In short, the branching of a tree at deterministic terms (such as in $\hat{t}$) has a different meaning than the branching at $\sqcup$ in $\hat{s}$. The set interpretation of the latter is quite natural and most authors do interpret it this way [82, 97, 23]. Unfortunately, the formalization of this intuition causes serious problems.

As the example suggests, we would like to define a mapping, Ψ: $\hat{W}_{\Sigma,\sqcup,\bot} \to \mathcal{P}(\hat{W}_{\Sigma,\bot})$ sending a (possibly infinite) term $\hat{t}$ to the set of its result trees, i.e., trees where $\sqcup$ has been replaced by the chosen branches, and such that the following diagram commutes:

$$\begin{tikzcd} & \hat{W}_{\Sigma,\sqcup,\bot} \arrow[dl, "\Psi"'] \arrow[dr, "\Psi_A"] & \\ \mathcal{P}(\hat{W}_{\Sigma,\bot}) \arrow[rr, "\mathcal{I}"] & & \mathcal{P}(A_\bot) \end{tikzcd}$$

Fig. 5.5

where: 1. $A_\bot$ is a continuous Σ-algebra and $\mathcal{P}(A_\bot)$ is the power set of its carrier. Notice that $A_\bot$ is deterministic since $\sqcup \notin \Sigma$.

2. $\mathcal{I}$: $\hat{W}_{\Sigma,\bot} \to A_\bot$ is the interpretation of the determinate terms in $A_\bot$. It is shown in [96] that such a continuous interpretation can be obtained as a unique extension of the interpretation of finite terms $W_{\Sigma,\bot} \to A_\bot$.

3. $\mathcal{I}$: $\mathcal{P}(\hat{W}_{\Sigma,\bot}) \to \mathcal{P}(A_\bot)$ is the pointwise extension of 2. to sets.

4. Ψ_A is a "direct interpretation" of terms involving $\sqcup$ in $\mathcal{P}(A_\bot)$ which should satisfy:

i) $\Psi_A(t) = \{ \mathcal{I}(t) \}$ for $t \in \hat{W}_{\Sigma,\bot}$

ii) $\Psi_A(t \sqcup s) = \Psi_A(t) \cup \Psi_A(s)$

iii) $\Psi_A(f(t_i)) = \{ f^A(a_i) \mid a_i \in \Psi_A(t_i) \}$

Equations 4.i-ii) require that Ψ_A interprets $\sqcup$ as the set of results obtained by performing all choices in the term, and iii) requires singular semantics of the arguments.

If there exists such a Ψ which is continuous then we can construct the lub in $\hat{W}_{\Sigma,\sqcup,\bot}$, map it on the set of the result trees in $\mathcal{P}(\hat{W}_{\Sigma,\bot})$ and interpret the elements of the latter in $A_\bot$, or, alternatively, interpret the equation using Ψ_A in $\mathcal{P}(A_\perp)$ and solve it there. Commutativity of the diagram and the equations 4. ensure that the results obtained in these two ways will be the same and, by the continuity of Ψ and $\mathcal{I}$, will yield lub in $\mathcal{P}(A_\perp)$. The continuity requirement on Ψ demands a definition of an ordering on $\mathcal{P}(\hat{W}_{\Sigma,\perp})$. Unfortunately, Broy reports the following result

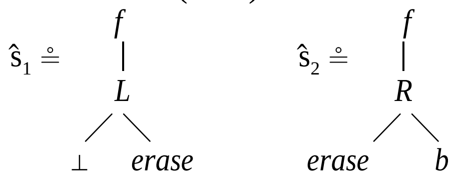
Proposition 5.6 [23]. There does not exist an ordering on $\mathcal{P}(\hat{W}_{\Sigma,\perp})$ such that Ψ is continuous.
□

Notice, that the ordering $\sqsubseteq$ on $\hat{W}_{\Sigma,\sqcup,\perp}$ is not flat. The problem here is the same as in defining an adequate ordering making power domains of non-flat domains cpos. We sketch two solutions of this problem. The first one modifies the structure $\mathcal{P}(\hat{W}_{\Sigma,\perp})$ so that an appropriate ordering can be defined. The second releases the continuity claim and, instead, applies more refined techniques of finding fixed points.

For the finite terms the situation is rather simple and equation 4.ii) will, for instance, make $\Psi(s)$ from example 5.4.a return the two expected trees $\hat{s}_1$ and $\hat{s}_2$. To envisage a general solution Boudol introduced [16, 97] *choice trees* for terms containing $\sqcup$. In addition to the chosen branches choice trees also keep a record of discarded branches.

Example 5.4.b

The choice trees for $s \circeq f(\perp \sqcup b)$ will be:



□

L(eft), R(ight) and *erase* are the new symbols added to the alphabet for indicating which of the branches of the original syntactic tree has been selected. The ordering on the choice trees is again $\sqsubseteq$. For $\hat{t}\in\hat{W}_{\Sigma,\sqcup,\perp}$ denote by $CT(\hat{t})$ the set of the choice trees of $\hat{t}$ and define the mapping ψ: $\hat{W}_{\Sigma,\sqcup,\perp} \to \mathcal{P}(\hat{W}_{\Sigma,L,R,erase,\perp})$ by $\psi(\hat{t}) = CT(\hat{t})$. This mapping is still not continuous (and it is shown, analogously to 5.6, that no ordering for the target can make it continuous). But a slight modification yields the desired continuity. Let $CT_{\Sigma,\perp} = \{(\hat{t},\hat{c}) \mid \hat{t}\in\hat{W}_{\Sigma,\sqcup,\perp}, \hat{c}\in CT(\hat{t})\}$ ordered by $(\hat{t},\hat{c})\sqsubseteq(\hat{t}',\hat{c}')$ iff $\hat{t}\sqsubseteq\hat{t}'$ and $\hat{c}\sqsubseteq\hat{c}'$. Order $\mathcal{P}(CT_{\Sigma,\perp})$ by the Egli-Milner extension [33] of $\sqsubseteq$, i.e., $\forall T,S\in\mathcal{P}(CT_{\Sigma,\perp})$

$$T \sqsubseteq_{EM} S \text{ iff } \forall tc\in T, \exists sc\in S : tc\sqsubseteq sc \wedge \forall sc\in S, \exists tc\in T : tc\sqsubseteq sc \qquad \text{(EM)}$$

Then the mapping Ψ, defined as $\Psi(\hat{t}) = \{(\hat{t},\hat{c}) \mid \hat{c}\in CT(\hat{t})\}$ is continuous.

The possible dissatisfaction with the fact that the intermediate structure introduced by the L, R and *erase* symbols had to be used, and that the final structure is not $\mathcal{P}(\hat{W}_{\Sigma,\perp})$ is resolved in [23]. Broy shows there that, although no continuous Ψ exists, one can find a continuous mapping into (an appropriate representation of) the Plotkin power domain [103] which factors through Ψ. Let $\mathcal{P}(A_\perp)$ denote the Plotkin power domain over $A_\perp$, $\mathcal{P}$ the mapping $\mathcal{P}(A_\perp) \to \mathcal{P}(A_\perp)$, and let $\mathcal{C}$ send sets of $A_\perp$ to their closures (i.e., for $B\in\mathcal{P}(A_\perp)$, $\mathcal{C}(B)$ contains least upper bounds and greatest lower bounds of, respectively. upward and downward directed, subsets of B). Then there exists a Ψ_A such that $\mathcal{P}\circ\Psi_A$ is continuous with respect to the Egli-Milner ordering, Ψ_A satisfies the equations 4., and such that the following diagram commutes

$$\begin{tikzcd}
& \hat{W}_{\Sigma,\sqcup,\perp} \arrow[dl, "\Psi"'] \arrow[dr, "\Psi_A"] & \\
\mathcal{P}(\hat{W}_{\Sigma,\perp}) \arrow[rr, "\mathcal{I}\circ\mathcal{C}"] \arrow[d, "\mathcal{P}"'] & & \mathcal{P}(A_\perp) \arrow[d, "\mathcal{P}"] \\
\mathcal{P}(\hat{W}_{\Sigma,\perp}) \arrow[rr, "\mathcal{I}\circ\mathcal{C}"] & & \mathcal{P}(A_\perp)
\end{tikzcd}$$

Fig. 5.7

Commutativity of the upper triangle of this diagram is the required Mezei-Wright-like result. One can interpret a term $\hat{t}\in\hat{W}_{\Sigma,\sqcup,\perp}$, using Ψ_A, directly in $\mathcal{P}(A_\perp)$ as a set of the elements of $A_\perp$ or interpret $\hat{t}$ as a set of trees over $\hat{W}_{\Sigma,\perp}$ which are then interpreted in $A_\perp$. Notice that all sets in the image of Ψ_A (and Ψ) are closed. This extends the notion of finite observability – if all finite approximations of an element s are in the set $\Psi_A(\hat{t})$ then so will s.

The construction of Ψ_A involves finding fixed points in $\mathcal{P}$(A), $\mathcal{S}$(A) (the Smyth power domain [117]) and taking the minimal function approximating the former and approximated by the latter. For the details of this construction, we refer the reader to [22, 23]. An extensive, both introductory and advanced, material on the power domains and continuity of the nondeterministic choice can be found in [106, 110, 118, 5, 99, 105, 3, 31, 18, 48].

The discussion above was concerned with singular (IO) semantics only. Detailed comparison of the Mezei-Wright-like results for IO vs. OI algebras can be found in [34, 35].

6. Reasoning Systems

As the variety of the proposals discussed so far indicates, the semantics of nondeterminism is certainly not a closed research topic. This is perhaps one of the reasons why relatively little work focuses on the reasoning about nondeterminism. Initially, one attempted to reduce such reasoning to reasoning about some deterministic equivalents. These attempts are the topic of 6.1. Only very recently some authors began to design systems for direct reasoning about nondeterminism. We give an example of such a system in 6.2. The following section 7 discusses the reasoning systems based on rewriting.

6.1. Reduction to determinism

The early approaches attempted to effect a reduction of the nondeterministic axioms to the semantically equivalent deterministic ones and to apply the standard forms of reasoning to the latter. In [121] such a translation is given for a specification language with equality interpreted as set equality. In order to specify nondeterministic operations one also allows *characteristic predicates* describing the result sets. These predicates, although a part of the specification, are not, strictly speaking, a part of the specification language. (They are similar to the "semantic functions" *loops* and *elem* from [21], or *breadth* from [18].)

Example 6.1

Binary nondeterministic choice

$\sqcup$: Nat×Nat→Nat

can be specified with the help of the predicate

$P_\sqcup$: Nat×Nat×Nat→Bool

$P_\sqcup(x,y,z)$ is to be true iff z is a possible result of $x\sqcup y$. So we define:

$P_\sqcup(x,y,z) \;=\; (x{=}z \vee y{=}z)$

□

The problem of reasoning with the nondeterminate terms arises because they do not have the *substitutivity property*. Variables refer to the elements of the semantic domain while nondeterminate terms denote sets of such elements. The characteristic predicate specification above is reasonable only because the variables x, y, z refer to single values and not sets (or arbitrary terms). Thus the substitution of $x\sqcup y$ for z, although consistent with the signature and syntactically correct, is not really meaningful since it yields

$P_{\sqcup}(x,y,x\sqcup y) \;=\; (x=x\sqcup y \vee y=x\sqcup y)$

which would make choice return always the first or always the second argument. In a sense, the predicates are at a different syntactic level of the specification than the (nondeterminate) terms. The formulae one reasons about are equations, and predicates are used only to make the following translation into the first order language possible.

Let us consider the equation:

$$x\sqcup y = y\sqcup x \qquad \text{(a)}$$

Equality is interpreted as set equality, so the translated determinate formula will be:

$$\forall z_1{:}[P_{\sqcup}(x,y,z_1) \Rightarrow \exists z_2{:}(P_{\sqcup}(y,x,z_2) \wedge z_1{=}z_2)] \wedge \forall z_2{:}[P_{\sqcup}(y,x,z_2) \Rightarrow \exists z_1{:}(P_{\sqcup}(x,y,z_1) \wedge z_1{=}z_2)] \qquad \text{(a}'\text{)}$$

It would be rather annoying to write the general translation pattern for equality, but the above example illustrates the point. It says: (a) holds iff for every possible result z_1 returned by $x\sqcup y$ there exists a result z_2 returned by $y\sqcup x$ such that $z_1{=}z_2$ and vice versa.

“Elementwise” reasoning can then be carried out in full first order logic. Denote the result of translating an equation e by $FOL(e)$ (for First Order Logic). All equational axioms e of the specification are translated into $FOL(e)$ and a formula a is valid iff $FOL(a)$ follows from the derived axioms. One can save some work by avoiding translation of deterministic terms and carrying them directly to the derived formulae. However, it is in general undecidable whether a given nondeterminate term is deterministic or not [18], so such an improvement would be limited to a syntactic check on the occurrences of nondeterminate subterms.

FOL will leave all logical symbols except for = unaffected, e.g., $FOL(\neg e) = \neg FOL(e)$, $FOL(e_1 \vee e_2) = FOL(e_1) \vee FOL(e_2)$, etc. Using this observation, [64] generalizes *FOL* to conditional equations. Let b be a Boolean expression in the specification language (not a characteristic predicate), e an equation. The translation of $b \Rightarrow e$ is defined as $FOL(b{=}\mathrm{T}) \Rightarrow FOL(e)$. For instance, take b to be $0 < 0\sqcup 1$, and consider the conditional formula:

$$0 < 0\sqcup 1 \;\Rightarrow\; e \qquad \text{(b)}$$

It will be translated as

$$[\ \forall z{:}(\ \mathrm{P}_{\sqcup}(0,1,z) \Rightarrow 0<z\ ) \wedge \exists z{:}0<z\ ] \;\Rightarrow\; \mathrm{FOL}(e) \qquad \text{(b}'\text{)}$$

All nondeterminate subexpressions of b are replaced with new variables ($0<z$) and these are bound in the characteristic predicates ($P_{\sqcup}(0,1,z)$) to be the results of the corresponding operations. Then the condition of (b′) is true if b holds for all possible results of its nondeterministic subexpressions (and there exists a value making b true – this, apparently superfluous, conjunct is obtained automatically by applying FOL to $b{=}\mathbb{T}$). The consequent is just the translation of the consequent of (b).

Since $0 \not< 0$ the antecedent of (b′) is false and so (b) will be considered true. This may be not quite what we intended when writing (b) since we could have been interested in making e hold only in the cases when $0\sqcup 1$ returns 1. The translation enforces a uniform interpretation of Boolean expressions – they are either true or false but not indeterminate. This reflects a general assumption made in [64] that Boolean expressions are, at most, weakly nondeterministic – for any argument (a single value) they can be evaluated nondeterministically but their result must be uniquely determined by the arguments. For instance, a Boolean operation $b(x)$ defined as $0<x$ would be legal, but the one defined as $0 \sqcup 1<x$ would not. The restriction is motivated by the fact that Boolean expressions are used there for handling exceptions, but it also indicates the limitations of the approach.

6.2. Calculus of nondeterminate terms

It is far from pleasing that starting with an equational specification, and being interested only in its equational consequences, one has to apply full first order logic. But it seems that the complexity of the equational logic of nondeterminism exceeds by far the complexity of the classical equational reasoning. Other systems which can deal with nondeterminism, especially variants of modal logics (temporal logic [83, 10, 71], dynamic logic [47, 36, 46]), are based on conceptual models which are much more complex than one would expect, and wish, for a logic of simple specifications with equations/inclusions.

An elegant contribution is that of unified algebras which, being based exclusively on Horn clauses, offer the possibility of taking the full advantage of the existing tools for reasoning about Horn specifications. The idea, however obvious, is only implicit in the presentation of unified algebras – the framework is based on the notion of institutions and deliberately abstracts from the peculiarities of the reasoning systems which might possibly be coupled with it.

As mentioned before, the main problem with nondeterminate terms is that they cannot be freely substituted for the (singular) variables. This introduces highly non-standard features into logics of nondeterminism. The only, to our knowledge, calculus for direct reasoning about nondeterminism which has been shown complete wrt. the multialgebraic semantics is given in [127, 126]. The calculus, NEQ, can be used for reasoning about disjunctive clauses where each clause C in NEQ is a set of atomic formulae (i.e., the ordering and multiplicity of the atomic formulae do not matter), written as $e_1, \ldots, e_n$. A clause is interpreted as the disjunction of its literals.

The atomic formulae e in NEQ are inclusions ($s \prec t$), equalities ($s \doteq t$), and inequalities ($t \# s$). Equalities are understood to mean *necessary* equality (i.e. the two terms *always* return the same result). $\vdash s \doteq t$ means not only that t and s are equal but also that they are deterministic, and $\vdash t \doteq t$ holds only for the deterministic terms. (Thus it is element equality and the symbol "$\doteq$" is used instead of "=" to emphasize the distinction w.r.t. standard equality). Inequalities are understood to mean *necessary* inequality (i.e. the two terms *never* return the same results). Variables are singular and a clause C is satisfied iff the formula $\forall x_i (e_1 \vee \ldots \vee e_m)$ is satisfied, where x_i are all variables in C. Equations and inclusions are called *positive* atoms, and inequations *negative* atoms. The latter act as negations of both equalities and inclusions. (A slightly different language where the negation of equality does not coincide with the negation of inclusion is used in [130].) For instance, a clause $\{ t \# s,\ p \prec r \}$ is thought of as the (equivalent) conditional formula $\neg(t \# s) \Rightarrow p \prec r$, stating that p is included in r whenever t intersects s. A clause with exactly one positive atom is a *Horn formula*, and Horn formulae with no negative atoms are *simple formula*.

The introduction of inequalities is motivated by the need of *binding* the applications of nondeterministic operations, and thus distinguishing between the terms treated as sets and the results of particular applications of terms corresponding to individuals. A formula $\vdash 0 \sqcup 1{=}0,\ 0 \sqcup 1{=}1$ says that $0 \sqcup 1$ is a set which is either equal to 0 or to 1, in other words, $0 \sqcup 1$ is not nondeterministic, merely underspecified. In order to express nondeterminism, one has to indicate that not the set but an arbitrary application of (element of the set) $0 \sqcup 1$ equals 0 or 1. This is expressed by the binding:

$$\vdash x \# (0 \sqcup 1),\ x \doteq 0,\ x \doteq 1. \tag{6.2}$$

(I.e., for any value x, x is either one of 0 and 1, or it is not a possible result of $0 \sqcup 1$.) Thinking of terms as sets, an inequality $\vdash \ldots s \# t \ldots$ is a short form for $\vdash \ldots z \in s, z \in t \ldots$, where z is a new variable, and the inequation in (6.2) corresponds requiring $x \in 0 \sqcup 1$ in the rest of the clause. The two inclusions $\vdash 1 \prec 0 \sqcup 1$ and $\vdash 0 \prec 0 \sqcup 1$ have the expected multialgebraic meaning which, together with the above disjunction, will force $0 \sqcup 1$ to be interpreted in all models exactly as the set $\{0,1\}$. Horn-formulae are a special case of clauses and the obvious restriction of the calculus below can be applied to this case. The rules of the calculus are as follows :

R1: a) $\vdash x \# y, x \doteq y$ b) $\vdash x \# t, x \prec t$ $x, y \in \mathcal{V}$

R2: $$\frac{\vdash C^x_t \quad ; \quad \vdash D, s \doteq t}{\vdash C^x_s, D}$$

R3: $$\frac{\vdash C^x_t \quad ; \quad \vdash D, s \prec t}{\vdash C^x_s, D}$$ x not in a right-hand side of $\prec$ in C

R4: $$\frac{\vdash C, s \preceq t \quad ; \quad \vdash D, s \# t}{\vdash C, D}$$ (CUT) ($\preceq$ being either $\doteq$ or $\prec$)

R5: $$\frac{\vdash C}{\vdash C, e}$$ (WEAK)

R6: $$\frac{\vdash C, x \# t}{\vdash C^x_t}$$ (ELIM) $x \in \mathcal{V} - \mathcal{V}[t]$, at most one x in C

Uppercase Latin letters C, D, denote single clauses and lowercase Latin letters a, e atomic formulae. Semicolon indicates a conjunction of clauses. E.g., if C is $\{c_1, \ldots, c_n\}$, D is $\{d_1, \ldots, d_m\}$, and a is an atomic formula, then "C ; D" denotes the conjunction of the two clauses, while "C, D" denotes the clause $\{c_1, \ldots, c_n, d_1, \ldots, d_m\}$. Similarly, "$C$, a" is the clause $\{c_1, \ldots, c_n, a\}$ and "C ; a" is the conjunction of C and $\{a\}$. C^x_t denotes C with t substituted for x.

A few comments regarding the rules may be in order.

R1 expresses the relation between $\#$ and equality and inclusion. Since variables x and y are individuals, the two rules correspond to, respectively, $x \neq y \vee x = y$, and $x \notin t \vee x \in t$. They also reflect the fact that '$\doteq$' is a partial equivalence relation and is reflexive only for variables.

R2 is a paramodulation rule allowing replacement of deterministic terms (in the case when $s \doteq t$ holds in the second assumption). In particular, it allows derivation of the standard substitution rule when the substituted terms are deterministic, and prevents substitution of nondeterministic terms for variables.

R3 allows "specialization" of a clause by substituting for a term t another term s which is included in t. The restriction that the occurrences of t which are substituted for don't occur in the right-hand side of $\prec$ in C is needed to prevent, for instance, the unsound conclusion $\vdash p \prec s$ from the premises $\vdash p \prec t$ and $\vdash s \prec t$.

$s \# t$ implies both negation of $s \doteq t$ and of $s \prec t$. R4 allows us to resolve these complementary atoms.

R5 allows one to weaken the premise clause by extending it with additional disjuncts.

R6 eliminates redundant bindings, namely those that bind an application of a term occurring at most once in the rest of the clause.

The main result concerning this calculus is its soundness and completeness:

Theorem 6.3 [127, 126]. For any specification SP: MMod(SP) $\models C$ iff SP $\vdash C$.
□

The tight relation between multimodels and functional models referred to in proposition 2.6 is further strengthened by the fact that the same theorem holds when we replace MMod by FMod.

7. Operational models and rewriting

Many authors approach the problem of reasoning about algebraic specifications using some form of rewriting. Rewriting systems have at least two features distinguishing them from the general form of reasoning discussed in this section: 1) they focus on (and are much more amenable to) automation and 2) they may be also seen as a way of giving the operational semantics to the specifications.

Nondeterministic operations have the inherent computational ingredient which deterministic functions lack: their result depends on the actual computation since nondeterministic decisions are made only when the program is executed. The power set constructions model this by considering *all possibilities* in one single model. An operational interpretation, on the other hand, incorporates the fact that nondeterminism is resolved during computation into the semantic structure.

Such structure is given by a *reduction system* which simulates the evaluation of terms (programs). Different interpretations of the objects being reduced (as strings, sequences, graphs) and different reduction strategies give rise to a variety of operational semantics.

7.1. Non-confluence and restricted substitutivity

The main idea is to simulate the computation by allowing a nondeterminate term to reduce to any of its possible result. For instance, the reduction rules for the binary choice will be:

$$x \sqcup y \rightarrowtail x \quad \text{and} \quad x \sqcup y \rightarrowtail y \tag{7.1}$$

This idea is applied in various operational semantics for the denotational models and λ-calculus. In [66, 67] the computation sequences (simulating sequences of nondeterministic choices), rather than the result sets, are introduced as the basis for a power domain construction. In [70] a similar semantics is shown equivalent to the operational semantics of bounded nondeterminism. [5] considers a generalization to unbounded nondeterminism and points out that power domains based on the computation sequences help solve the problem of noncontinuity of unbounded nondeterministic choice. In [3, 105] noncontinuity is tackled by the generalization of the semantics of the iterative construct to fixed points over transfinite ordinals (according to the suggestion from [99]), and the resulting Plotkin and Smyth power domains for countable nondeterminism are shown equivalent to the respective operational semantics. An operational interpretation of λ-calculus extended with the choice operator can be obtained by appropriate modifications of the β-rule. Such modifications, leading to both plural and singular semantics, are discussed, for instance, in [49, 73, 4]. We will base the following discussion on the terminology of the systems for term rewriting which constitute a natural counterpart to the algebraic specifications.

Since x and y in (7.1) may be arbitrary terms, in particular both may be in *normal form* (not reducible by the rules of the system) rewriting with nondeterminism will be, typically, *non-confluent* [24, 59, 57, 97, 4, 87]. One may simply accept it as a fact of life – after all, a nondeterministic computation is precisely one which can return several results, and each rewrite represents one among them [59, 4, 87]. Nevertheless, one may try to show some desirable properties of the resulting system. For instance, in [4] the authors show that their rewriting system is confluent for all terms not involving $\sqcup$, and that for all terms the possible rewrites constitute exactly the set prescribed by the independently defined model. In the cases when the semantics of a specification is defined also by non-operational means the latter correspondence between the set obtained by all possible rewritings of a term and the set assigned to this term by the static semantics is, of course, an obligation. Inclusion of the former in the latter yields soundness and the opposite inclusion completeness of the reasoning by rewriting.

An exception to the schema (7.1) is the work of Kaplan [63]. It actually forbids rules like these, except for the one rule of distributivity of $\sqcup$ over function application (JD from 3.1 oriented from left to right). Instead, terms in normal form may involve $\sqcup$. For instance, one may define an operation f by the rule $f \rightarrowtail a\sqcup b$. If a and b are irreducible $a\sqcup b$ will be the normal form of f. Rewriting is then performed modulo the equations for associativity and commutativity of $\sqcup$ with the rules corresponding to other join axioms (JI and JE from 3.1). This leads to the possibility of obtaining unique normal forms for the nondeterminate terms (bounded nondeterminism) and the conditions for such systems to be confluent are given. An interesting property of such confluent systems is that their initial models are isomorphic to the intuitive multialgebra models with $\sqcup$ interpreted as set union.

One of the issues raised by virtually any work on rewriting is the distinction between the singular and the plural semantics of parameter passing [34, 35, 49, 4, 73]. The need to decide when the arguments of the terms under evaluation are to be reduced makes this distinction natural, not to say unavoidable. The two evaluation strategies are present already in the rewriting of deterministic terms [43, 98]. Most generally, the singular semantics requires the arguments to be evaluated before the enclosing operation call (IO), while the plural one postpones their evaluation until the enclosing term has been reduced in its entirety (OI). This gives the two substitution rules:

$$f(x)\ [x/t]_{\mathrm{IO}} \rightarrowtail \bigcup\{\ f(s) \mid s{\in}t\ \} \qquad \text{(IO)}$$

$$f(x)\ [x/t]_{\mathrm{OI}} \rightarrowtail f(t) \qquad \text{(OI)}$$

We have adopted the rather informal notation $f \rightarrowtail \{...\}$ to indicate that f can be rewritten to any element of the set on the RHS of $\rightarrowtail$, and $s{\in}t$ to say that s is (one of) the normal forms of t. For instance, let $f(x) \rightarrowtail x{+}x$ and $t \doteq 0{\sqcup}1$. Evaluation of $f(t)$ under IO requires that we first find all the rewrites of $0{\sqcup}1$, and will give us $f(x)[x/0{\sqcup}1]_{IO} \rightarrowtail \{f(0), f(1)\} \rightarrowtail \{0,2\}$. OI, on the other hand will yield $f(x)[x/0{\sqcup}1]_{OI} \rightarrowtail f(0{\sqcup}1) \rightarrowtail 0{\sqcup}1{+}0{\sqcup}1 \rightarrowtail \{0,1,2\}$.

OI is a much simpler mechanism than IO. The latter requires that the argument t be first reduced to (some of) its normal forms s before $f(t)$ can be evaluated. Furthermore, all such normal forms must be deterministic in order for IO to yield the singular semantics! For if t can be reduced to a nondeterminate normal form s then $f(x)[x/t]_{IO}$ will be reduced to a set $\{...,f(s),...\}$ and then to $\{...,s{+}s,...\}$. Thus, having the singular semantics in mind, one must, in addition to designing an IO rewriting system, make sure that the system, as well as the specification, satisfy some additional conditions. Such conditions are given by Hußmann in [59, 58]. The rewriting system must satisfy the condition that any substitution of a term s for a variable x be guarded by the prior proof that s is deterministic, $DET(s)$. The responsibility of the specifier is to ensure that all terms do have a deterministic normal form, i.e.:

$$\forall t{\in}W_{\Sigma}\exists s{\in}W_{\Sigma} : t \overset{*}{\rightarrowtail} s \wedge \mathrm{DET}(s) \qquad \text{(DET-completeness)}$$

where $\overset{*}{\rightarrowtail}$ is the transitive closure of $\rightarrowtail$. One more condition is needed to ensure the singular semantics. Any (deterministic) result obtained from a term $f(t)$ must be a result obtained by making some choices during the evaluation. The rule $f(0\sqcup 1) \rightarrowtail c$ does not determine singularity of the argument passed to f. If c is to be produced in the singular semantics from $f(0\sqcup 1)$ it must be the result of either $f(0)$ or $f(1)$. This requirement, termed *DET-additivity*, is ensured if the specification with the rewriting rules R satisfies two conditions:

$$1.\ \forall\ l \rightarrowtail r \in R \ :\ l \doteq f(t_1,...,t_n) \wedge \forall i\ .\ \mathrm{DET}(t_i)$$

$$2.\ \forall\ f(t_1,...,t_n)\ :\ \mathrm{DET}(f(t_1,...,t_n)) \Rightarrow \mathrm{DET}(t_i) \qquad \text{(DET-additivity)}$$

Besides the rules with single variable in the LHS, 1. forbids the rules like $f(0\sqcup 1) \rightarrowtail c$ and requires that one write instead the rules $f(0) \rightarrowtail c$ and $f(1) \rightarrowtail c$. Analogous requirements of *left* $\sqcup$*-freeness* is made in [63]. 2. says that an operation f with the nondeterministic arguments is capable of returning a set of possible results. This is again the condition analogous to one made by Kaplan in [63]. Compared with the latter work the system of Hußmann's [59] is much less restrictive since in [63] it is also required that rules be left-linear, i.e., contain no multiple occurrences of a variable in LHS. The rewriting system of Hußmann is shown to be sound and (weakly) ground complete with respect to the model class (MMod,MHom) of DET-complete and DET-additive specifications. The calculus of section 6.2 is sound and complete for specifications without the restrictions to DET-completeness and -additivity.

What makes nondeterministic OI so much simpler than IO is the fact that the models of the former are essentially deterministic power algebras. Hence, the variables in OI refer to sets which are just individuals of the carrier. This is succinctly expressed in:

Proposition 7.2 [34]. OI substitution is associative, i.e.:

$$f(..x..y..z..)\ ([x/t][y/s])[z/p]_{\mathrm{OI}} = f(..x..y..z..)\ [x/t]([y/s][z/p])_{\mathrm{OI}}.$$

□

A simple example shows that this is not the case for the IO substitutions:

$$x + y\ ([y/x][x/0\sqcup 1])\ [x/0\sqcup 1]_{\mathrm{IO}} = x + y\ [y/0\sqcup 1]\ [x/0\sqcup 1]_{\mathrm{IO}} = \{0,1,2\} \neq$$
$$x + y\ [y/x]\ ([x/0\sqcup 1][x/0\sqcup 1])_{\mathrm{IO}} = x + x\ [x/0\sqcup 1]\ [x/0\sqcup 1]_{\mathrm{IO}} = x + x\ [x/0\sqcup 1]_{\mathrm{IO}} = \{0,2\}$$

The lack of associativity of IO substitutions is the reason for the complications of the rewriting systems referred to above. The problem does not arise if there is always at most one occurrence of a variable to be substituted for or if the substituted terms are deterministic.

Proposition 7.3 [34]. If
1. there is at most one occurrence of the variable to be substituted for, or
2. any substituted term is deterministic then IO substitution is associative.

□

1. is reflected in the systems requiring *linearity*, i.e., forbidding multiple occurrences of a variable, which is a very strong restriction ([63], the introductory system in [58]), and 2. in the systems, like Hußmann's, which require a proof of determinacy for the substituted terms. (The calculus from subsection 6.2 takes care of both cases: the restriction on the rule R7 corresponds to 1, and the second premise in rule R2 to 2. In [130, 131] it is also shown that a complete extension of the calculus to plural arguments requires only the addition of the standard, unrestricted substitution rule for plural variables.)

Of course, one of the main differences between IO and OI strategies concerns, as in deterministic rewriting, the termination properties. Since the former requires the evaluation of all arguments before the evaluation of a call, it will diverge in many cases in which the latter will compute a well defined result. One of the numerous variations on the plural/singular vs. OI/IO distinctions is proposed in [4]. It combines the singular semantics of parameter passing with the preferable termination properties of the OI evaluation strategy by means of *sharing*. The idea is to bind (share) the arguments and postpone their evaluation until reduction of the term requires a value of the argument (lazy evaluation). The language (of λ-expressions, actually, but we stay with our notation) is extended with the binding construct $\{x/t\}$ which is performed before substitution:

$$f(x)\ [x/t]_\beta \rightarrowtail f(x)\ \{x/t\} \tag{1}$$

The rest of the system takes then care of rewriting f and t as far as possible, e.g.,

$$\begin{array}{r} \text{if } f(x) \rightarrowtail f'(x) \text{ then } f(x)\{x/t\} \rightarrowtail f'(x)\{x/t\} \\ \text{if } t \rightarrowtail t' \text{ then } f(x)\{x/t\} \rightarrowtail f(x)\{x/t'\} \end{array} \tag{2}$$

"As far as possible" is defined with the help of the *critical set*, *CR*, of a term. $CR(t)$ is empty whenever t can be rewritten and contains the variable x if further rewriting of t demands x to be evaluated. The rule

$$\text{if } x \in CR(f(x)) \text{ then } f(x)\{x/t\} \rightarrowtail f(x)[x/t] \tag{3}$$

performs then the usual substitution $[x/t]$ of t for all x in $f(x)$. By the rules (2), t in the moment of such substitution will be in normal form.

7.2. Rewriting Logic

A very powerful and general theory of nondeterministic (and concurrent) rewriting called *Rewriting Logic* was designed by José Meseguer in [87, 86].

As we have seen, purely equational specifications of nondeterminism do not yield intuitively plausible results: Defining choice by $x \sqcup y = x$, and $x \sqcup y = y$ gives $x = y$. On the other hand, the definition $x \sqcup y = x \vee x \sqcup y = y$ does not have an initial model. This motivates the use of inclusion (3.2–3.4) or rewrite rules as above. The language of rewriting logic extends the equational language by allowing also *conditional* rewrite rules. Equations may be considered two-way rewrite rules, but the idea is to simplify the exposition by rewriting in equivalence classes modulo the congruence induced by the equations and to use rewrite rules for nondeterministic terms only. The generalization of the rewriting systems for equational theories lies in allowing conditional rules and labeling the rules (we discuss this point in a moment). A rewrite system S is given by a 4-tuple (Σ, E, L, R) of signature, equations, labels, and rewrite rules labeled by L.

Example 7.4.a

For illustration, we will use the following system S for the natural numbers with binary choice

Σ:	0:		→ Nat
	succ:	Nat	→ Nat
	+:	Nat×Nat	→ Nat
	⊔:	Nat×Nat	→ Nat

E: 1. $0+x = 0$ 2. $x+y = y+x$
3. $succ(x)+succ(y) = succ(succ(x+y))$
4. $x \sqcup y = y \sqcup x$

L: $\{ r1, r2 \}$

R: $r1: x \sqcup y \rightarrowtail x$ $r2: x \sqcup y \rightarrowtail y$

□

Rewriting logic
The rewriting logic (*RL*) is the following set of rules ([*t*] denotes the equivalence class of *t* under the congruence induced by *E*; $t_i,s_i,w_i,v_i \in W_{\Sigma,X}$):

RL1. Reflexivity $\quad [t] \rightarrowtail [t]$

RL2. Congruence $\quad \dfrac{[t_i] \rightarrowtail [s_i]}{[f(t_i)] \rightarrowtail [f(s_i)]}$

RL3. Replacement $\quad \dfrac{[w_i] \rightarrowtail [v_i]}{[t(w_i)] \rightarrowtail [s(v_i)]}$ $\quad$ for each rule r: $t(x_i) \rightarrowtail s(x_i)$ in R

RL4. Transitivity $\quad \dfrac{[t_1] \rightarrowtail [t_2] \quad [t_2] \rightarrowtail [t_3]}{[t_1] \rightarrowtail [t_3]}$

This *is* the whole logic and its simplicity and intuitive appeal call for no further comments. (We have simplified the rule RL3 which, in general, involves conditionals.) Like in the most systems discussed above, a proof in RL will produce a possible result of the argument, for instance, RL1 and RL3 will give $[x \sqcup y + z] \rightarrowtail [x+z]$ (using *r1*), and $[x \sqcup y + z] \rightarrowtail [y+z]$ (using *r2*).

The model $\mathcal{T}_S(X)$
A model of a system S is constructed as a category $\mathcal{T}_S(X)$ with the objects being the equivalence classes [*t*] for $t \in W_{\Sigma,X}$, and the morphisms the equivalence classes of the terms representing proofs in RL. In order to identify appropriate proof terms (and possibly distinguish two morphism with the same source and target) the labels of the rules applied are used. The morphisms are generated using the rules "simulating" the proof rules of RL which, in addition, attach the appropriate label to each morphism:

1. Identities $\quad [t]: [t] \rightarrowtail [t]$

2. Σ-structure $\quad \dfrac{\alpha_i : [t_i] \rightarrowtail [s_i]}{f(\alpha_i) : [f(t_i)] \rightarrowtail [f(s_i)]}$

3. Replacement $\quad \dfrac{\alpha_i : [w_i] \rightarrowtail [v_i]}{r(\alpha_i) : [t(w_i)] \rightarrowtail [s(v_i)]}$ $\quad$ for each rule r: $t(x_i) \rightarrowtail s(x_i)$ in R

4. Composition $\quad \dfrac{\alpha : [t_1] \rightarrowtail [t_2] \quad \beta : [t_2] \rightarrowtail [t_3]}{\alpha;\beta : [t_1] \rightarrowtail [t_3]}$ $\quad$ (writing composition from left to right)

A morphism γ: $[t] \rightarrowtail [s]$ corresponds then to a proof (term) which, by applying the rules γ, starts with [*t*] and produces [*s*]. To make a category out of this one needs to ensure the associativity of compositions and to make the morphisms [*t*] into the identities on the objects [*t*]. The following axioms take care of imposing the required properties on the morphisms:

(assoc) $\quad$ for all α, β, γ $\;(\alpha;\beta);\gamma = \alpha;(\beta;\gamma)$
(ident) $\quad$ for all α: $[t] \rightarrowtail [s]$ $\quad \alpha;[s] = \alpha$ and $[t];\alpha = \alpha$

Some additional axioms make sure that the appropriate morphisms will be identified. For instance, axiom 4. makes $[0 \sqcup 1] = [1 \sqcup 0]$, and hence the morphisms *r1*([0],[1])*:* $[0 \sqcup 1] \rightarrowtail [0]$, and *r2*([1],[0])*:* $[0 \sqcup 1] \rightarrowtail [0]$ should be identified. In particular, all equations from *E* should be valid in the model category, so for every equational axiom $t(x)=s(x)$ and all α we require that the morphisms $t(\alpha)$ and $s(\alpha)$ be equal. This will make, for instance, +([0],[*x*]) equal to the identity morphism [*x*] for all *x*.

Example 7.4.b

A small piece of the category $\mathcal{T}_S(X)$ for S from example 7.4.a is shown below:

$$\begin{tikzcd}
 & & {[succ(0\sqcup 1)]} \arrow[loop above] \arrow[dll, "succ(r1([0],[1]))"'] \arrow[ddrr, "succ(r2([0],[1]))"] & & \\
{[0\sqcup 1]} \arrow[loop above] \arrow[d, "r1([0],[1])"'] \arrow[dr, "r2([0],[1])"] & & & & {[1\sqcup 2]} \arrow[loop above] \arrow[d, "r2([1],[2])"] \arrow[dlll, "r1([1],[2])"] \\
{[0]} \arrow[loop below] & {[1]} \arrow[loop below] & & & {[2]} \arrow[loop below]
\end{tikzcd}$$

□

Note that the object [$succ$(0⊔1)] is not the same as [1⊔2]. This reflects the ability of the model to distinguish between plural semantics (which is default) from the singular which would require the additional distributivity axiom $succ(x\sqcup y) = succ(x)\sqcup succ(y)$.

The last thing is to interpret the Σ-structure in $\mathcal{T}_S(X)$. Two final identifications are:

(comp) for all f, α_i, β_i $\quad f(\alpha_i;\beta_i) = f(\alpha_i);f(\beta_i)$

(ident) for all f, t_i $\quad f([t_i]) = [f(t_i)]$

With these equations, every $f\in\Sigma$ determines a functor $f_{\mathcal{T}_S}$: $\mathcal{T}_S(X)^n \rightarrow \mathcal{T}_S(X)$ (where n is the arity of f). For instance, $succ$ will determine the functor which sends every object [t] to [$succ$(t)], and every morphism, like $r1$([0],[1]), to the respective morphism $succ$($r1$([0],[1])).

The category of S-models

$\mathcal{T}_S(X)$ is just one among the possible models of S. For a category C to be an S-model it must satisfy all the axioms of S. Intuitively this means that

1. for every t=$s \in E$, the functors t_C and s_C induced in C should be identical,
2. for every $[t] \rightarrowtail [s] \in R$, there must exist a corresponding arrow in C.

Since t and s may involve variables, 2. is formalized as the requirement of the existence of a natural transformation between the respective functors t_C and s_C. Finally, the notion of a morphism of S-models is defined as an S-functor which preserves the Σ-structure and the natural transformations corresponding to each rule of S. This gives a category S-$\mathcal{CAT}$ of all models of S. One of the central results about these constructions is:

Theorem 7.5 [87]. The functor S-$\mathcal{CAT} \rightarrow \mathcal{SET}$ sending an S-model to the set of its objects has the left adjoint which sends each set X to $\mathcal{T}_S$(X).
□

The first thing one obtains from this theorem is an easy proof of completeness of RL with respect to S-$\mathcal{CAT}$ (of course, RL is also sound for this class). It also suggests a generalized initial semantics for rewrite systems since any system S has now the initial model $\mathcal{T}_S$. The list of various approaches which can be subsumed under rewriting logic is quite impressive. We only mention the link to the classical initial algebras.

Given a standard rewriting system S one can construct the initial model for S provided it is confluent. The present construction assigns an initial model to any S system – also the ones which involve nondeterminism in the form of non-confluent rewrite rules (or are not confluent for other reasons). If $\mathcal{S}$ happens to determine an initial algebra $T_{\mathcal{S}}$, or, more generally, if we are interested in the initial model $T_{\mathcal{S}}$ of the equational theory corresponding to $\mathcal{S}$, then $T_{\mathcal{S}}$ may be obtained as a quotient of $\mathcal{T}_{\mathcal{S}}$. The equational theory for $\mathcal{S}$ is the one where all $\rightarrowtail$ have been turned into =. This means that all we have to do is to impose such equalities on the morphisms (representing $\rightarrowtail$) in $\mathcal{T}_{\mathcal{S}}$. We unlabel all morphisms and make $[t]$=$[s]$ whenever there is a morphism $[t] \rightarrowtail [s]$. The image of $\mathcal{T}_{\mathcal{S}}$ under this transformation will be $T_{\mathcal{S}}$. In fact, the transformation is a functor and it will send $\mathcal{S}$-$\mathcal{CAT}$ onto the class Mod($\mathcal{S}$) of ordinary Σ-models of the equational theory of $\mathcal{S}$.

7.3. Reasoning and rewriting with sets.

The rewriting relation $t \overset{*}{\rightarrowtail} s$, interpreted in the equational case as the equality of t and s, has been redefined for the nondeterministic rewriting to mean that t is *possibly* equal to s (or that $s \prec t$). As we have seen, an immediate consequence is a lack of confluence. Since $\prec$ is not an equivalence relation, this poses a more general question of rewriting in the presence of such relations. In particular, inclusion is just one among the common set-relations. Thus reasoning about nondeterminism becomes closely related with reasoning with sets (as observed in [129]) – an area of increasing importance, especially in logic and constraint programming [114, 32, 61, 120]. The suggested extensions go in the direction of *rewriting atoms* – pairs of terms annotated with an appropriate relational symbol – rather than rewriting of terms.

The authors of *bi-rewriting* [72] propose the use of two kinds of relations: $\overset{\subseteq}{\rightarrowtail}$ and $\overset{\supseteq}{\rightarrowtail}$. To prove $t \prec s$, one will try to rewrite t to a term u replacing it by "bigger" terms along the relation $\overset{\subseteq}{\rightarrowtail}$. Simultaneously, one tries to rewrite s to u replacing it by "smaller" terms along the relation $\overset{\supseteq}{\rightarrowtail}$. Existence of such a u (which need not be unique) proves the inclusion. The use of two relations gives more flexibility in applying different simplification orderings because now the ordering of terms need not coincide with their inclusions $\prec$.

A related approach from [69] introduces a language for specifying the relations: $\prec$, $\frown$, and = which, interpreted in multialgebras, correspond to inclusion, intersection and identity of 1-element sets. The associated reasoning is based on the replacement and compositional properties of these relations. A sound and refutationally ground-complete system for the language of sequents over such atomic relations is given. Rewriting is done using the corresponding relations: $\overset{\subseteq}{\rightarrowtail}$, $\overset{\supseteq}{\rightarrowtail}$, $\overset{\frown}{\rightarrowtail}$, and $\overset{=}{\rightarrowtail}$. In particular, in the course of a rewriting proof the predicates may change according to the laws of their composition, yielding all the time the strongest possible relation between the active terms. For instance, two rewriting steps $t \overset{\supseteq}{\rightarrowtail} s$ and $s \overset{\subseteq}{\rightarrowtail} u$ will lead to a derivation of $t \overset{\frown}{\rightarrowtail} u$, while the steps $t \overset{\supseteq}{\rightarrowtail} s$ and $s \overset{=}{\rightarrowtail} u$ will yield $t \overset{\supseteq}{\rightarrowtail} u$.

The above works extend the equational rewriting generalizing several classical notions such as overlapping rules, critical pair, confluence. In particular, the notion of unique normal form becomes less central. Instead, one has to invent a search strategy which finds appropriate atoms/terms for chaining proofs (such as the term u in $s \overset{\supseteq}{\rightarrowtail} u$ and $t \overset{\subseteq}{\rightarrowtail} u$ needed to complete the proof of $t \prec s$).

8. Summary

Nondeterminism poses many problems which cannot easily be incorporated into the framework of deterministic specifications. The formalization of the intuition that nondeterministic operations generate sets of possible results, reflected in the power set structures, presents serious difficulties. The proposed generalizations of the classical results such as those on initiality, continuity, and equational reasoning to the power set structures have only limited validity. The most successful proposals are rather sophisticated and apply to non-standard constructions. They are often very general, addressing other problems besides that of nondeterminism.

The fundamental difficulty seems to lie in the inherently computational character of nondeterminism which confronts the attempts to enclose it in a single, preferably initial, model. This difficulty may be expressed by the slogan that "choice is not a constructor." An occurrence of the syntactic operation $\sqcup$ in specifications will typically be reflected in the initial model by the occurrence of the respective choice terms, such as $a \sqcup b$. They, however, represent a kind of "intuitive junk" because choice is not supposed to introduce any new elements into the model. On the other hand, the intended meaning cannot be covered by the purely equational language, which corrupts the nature of the choice operator and reduces it to a deterministic function.

The solution comes in some form of the "is a possible result" predicate. It is present in almost all approaches as inclusion, partial ordering or rewrite rules. In cases when the initial semantics can be obtained – and these are the cases when the specifications are restricted to Horn theories – such a predicate corresponds to the intuitive meaning of the nondeterministic operations.

For multialgebras, the initial semantics can be obtained only in particular cases. To guarantee its general applicability one has to introduce some partial order models. Such models, however, are based on blurring the distinction between individuals and sets. In particular, they give the predicate "is a possible result of" a mathematically simpler, but intuitively less obvious, meaning than the set based models do.

Loose semantics of nondeterminism has not received much attention. The restriction to Horn theories, even with the "is a possible result" predicate, have more undesirable consequences for such semantics than in the deterministic case. The general models of determinate specifications introduce junk (undenotable elements), but the behavior of the operations applied to such elements is still controlled by the axioms. This is no longer the case when we move to the nondeterministic specifications. The axioms $x \prec x \sqcup y$ and $y \prec x \sqcup y$ specify only the lower bound on the admitted nondeterminism of the choice operator. In the general power set models, the interpretation of nondeterministic ground terms such as $0 \sqcup 1$ will be allowed to return *any* imaginable (and unimaginable) results besides the two prescribed by such a specification. Thus junk will appear, so to speak, not only next to non-junk but also in the middle of it. (This does not apply to the unified algebras where the interpretation of choice as join excludes such situations.)

The computational character of nondeterminism finds its natural expression in the operational models. At the syntactic level it is reflected in the fact that nondeterminate terms do not denote unique values, and consequently violate the requirements of referential transparency and substitutivity. This leads to an operational approach not only in the semantics, but also in the reasoning. With the exception of the rewriting logic, this indicates why one had to await completeness results of the proposed reasoning systems for so long. Also, such systems usually describe the result set of a term not by a single formula but by a *set* of formulae – one for each possible result. Hence, a deduction of the result set of a term requires derivations of all such formulae. The reasoning must not only consider the syntax of the formulae involved but also ensure that all possible rules have been applied. However, this deficiency does not occur in the calculus from 6.2.

With respect to the last point, as well as to the deficiencies of Horn-specifications, disjunctive axioms provide significant help. However, such axioms do not admit initial models and introduce a few extra complications in the reasoning systems.

An important issue that has not been addressed in this paper concerns the notion of *implementation* of nondeterministic data types. The reason for this omission is the almost complete absence of any published results which, we suppose, reflects the absence of active research in that area. We may mention two works [107, 132] which both point in the same direction: In the context of data refinement it seems necessary to distinguish underspecification from nondeterministic operations by relaxing the referential transparency (unrestricted substitutivity) requirement with respect to the latter. (This relaxation corresponds to the restricted substitutivity in the logic discussed in section 6.2.) Mere underspecification excludes the possibility of verifying implementations which naturally would be considered correct and plausible.

Although there already exists an extensive literature on various aspects of nondeterminism in an algebraic setting, it does not seem that the research has established some consensus similar to the one discernible in the classical specifications of deterministic data types. Such a consensus – concerning not the best and only formalism, but the relative importance and applicability of different formalisms – is probably needed before the questions about implementation and structured specification of nondeterministic data types can receive closer attention.

9. References

[1] C. Aarts, R. Backhouse, P. Hoogendijk, E. Voermans, J. Woude, "A Relational Theory of Datatypes", 1992, In preparation.

[2] L. Aceto, R. DeNicola, A. Fantechi, "Testing Equivalencies for Event Structures", in *Mathematical models for the semantics of parallelism*, LNCS, vol. 280, Springer, 1986.

[3] K.R. Apt, G.D. Plotkin, "A Cook's tour of countable nondeterminism", in Proceedings of *8th International Colloquium on Automata, Languages and Programming,* LNCS vol. 115, Springer, 1981.

[4] E. Astesiano, G. Costa, "Sharing in Nondeterminism", in *Automata, Languages and Programming, Sixth Colloquium*, LNCS, vol. 71, Springer, 1979.

[5] R.J. Back, "Semantics of Unbounded Nondeterminism ", in *Automata, Languages and Programming*, LNCS, vol. 85, Springer, 1980.

[6] R.J.R. Back, J. von Wright, "Duality in Specification Languages: A Lattice-theoretical Approach", *Acta Informatica*, vol. 27, 583-625, 1990.

[7] J.W. de Bakker, "Semantics and Termination of Nondeterministic Recursive Programs", *Automata, Languages and Programming, Edinburgh*, 435 - 477, 1976.

[8] F. Bauer, *The Munich Project CIP: The Wide Spectrum Language CIP*, LNCS vol. 183, Springer, 1985.

[9] H. Bekic, *Programming Languages and Their Definition*, vol. 177, Springer Verlag, 1984.

[10] M. Ben-Ari, Z. Manna, A. Pnueli, "The temporal logic of branching time", in Proceedings of *Eighth Annual Symposium on Principles of Programming Languages,* 1981.

[11] J.A. Bergstra, J.W. Klop, *An Abstraction Mechanism for Process Algebras,* Tech. Rep. IW 231/83, Dept. of CS, Mathematisch Centrum, Amsterdam, 1983.

[12] J.A. Bergstra, J.V. Tucker, "Initial and final algebra semantics for data type specifications", *SIAM Journal on Computing*, vol. 12, 366-387, 1983.

[13] J.A. Bergstra, J.W. Klop, "Algebra of communicating processes", *Proceedings of CWI Symposium on Mathematics and CS,* 89 - 138, Oct. 6-7 1986.

[14] J.A. Bergstra, J.V. Tucker, "Algebraic specifications of computable and semicomputable data types", *Theoretical Computer Science*, vol. 50, 137-181, 1987.

[15] S.L. Bloom, R. Tindell, “Compatible orderings on the metric theory of trees”, *SIAM J. Comput.*, vol. 9, no. 4, 683-691, November 1980.

[16] G. Boudol, *Calculus maximaux et semantique operationnelle des programmes non deterministes*, Ph.D. thesis, University of Paris, 1980.

[17] M. Broy, W. Dosch, H. Parsch, P. Pepper, M. Wirsing, “Existential quantifiers in abstract data types”, in *Automata, Languages and Programming, Sixth Colloquium*, LNCS, vol. 71, Springer, 1979.

[18] M. Broy, R. Gnatz, M. Wirsing, “Semantics of Nondeterministic and Noncontinuous Constructs”, LNCS, vol. 69, Springer, 1980, pp. 553 - 392.

[19] M. Broy, M. Wirsing, *Initial versus Terminal Algebra Semantics for Partially Defined Abstract Types*, Tech. Rep. TUM-I 8018, Technische Universität München, December 1980.

[20] M. Broy, M. Wirsing, “Programming Languages as Abstract Data Types”, *Lille Colloque 80*, 1980.

[21] M. Broy, M. Wirsing, “On the Algebraic Specification of Nondeterministic Programming Languages ”, in *CAAP'81*, LNCS, vol. 112, Springer, 1981, pp. 162 - 179.

[22] M. Broy, “Fixed point theory for communication and concurrency”, in *IFIP TC2 Working Conference on Formal Description of Programming Concepts II,* North-Holland, 1983, pp. 125-147.

[23] M. Broy, “On the Herbrand Kleene universe for nondeterministic computations”, in *Proc. MFCS’84*, LNCS, vol. 176, Springer, 1984.

[24] M. Broy, “A Theory for Nondeterminism, Parallelism, Communication and Concurrency”, *TCS*, vol. 45, 1986.

[25] A.K. Chandra, “Computable Nondeterministic Functions”, *19th Annual Symposium on Foundations of Comp.Sci.*, 1978.

[26] W. Clinger, “Nondeterministic call by need is neither lazy nor by name”, *Proc. ACM Symp. LISP and Functional Programming*, 226-234, 1982.

[27] P.M. Cohn, *Universal Algebra*, Mathematics and Its Applications vol. 6, D.Reidel Publishing Company, 1965.

[28] O.J. Dahl, *Verifiable Programming*, Prentice Hall, 1992.

[29] E.W. Dijkstra, “Guarded Commands, Nondeterminacy and Formal Derivation of Programs”, *CACM*, vol. 18, 1975.

[30] E.W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, 1976.

[31] E.W. Dijkstra, “A simple fix-point argument without the restriction to continuity”, in *Control Flow and Data Flow*, Comp. and System Sciences, Springer, 1984.

[32] A. Dovier, E. Omodeo, E. Pontelli, G.F. Rossi, “Embedding Finite Sets in a Logic Programming Language”, LNAI, vol. 660, Springer, 1993, pp. 150-167.

[33] H. Egli, *A mathematical model for nondeterministic computations,* ETH, 1975.

[34] J. Engelfriet, E.M. Schmidt, “IO and OI. 1”, *Journal of Computer and System Sciences*, vol. 15, 328-353, 1977.

[35] J. Engelfriet, E.M. Schmidt, “IO and OI. 2”, *Journal of Computer and System Sciences*, vol. 16, 67-99, 1978.

[36] Y.A. Feldman, D. Harel, “A probabilistic dynamic logic”, in Proceedings of *Fourteenth Annual ACM Symposium on Theory of Computing* , 1982.

[37] J.A. Fernández, J. Minker, “Bottom-Up Evaluation of Hierarchical Disjunctive Dedutcive Databases”, *Logic Programming, Proc. of the Eight Int. Conf.*, 1991.

[38] R.W. Floyd, “Nondeterministic algorithms”, *Journal of the ACM*, vol. 14, no. 4, 1967.

[39] R.J. van Glabbeek, *Bounded nondeterminism and the approximation induction principle in process algebra*, Tech. Rep. CS-R8634, Centrum voor Wiskunde en Informatica, Amsterdam, 1986.

[40] J.A. Goguen, J. Thatcher, E. Wagner, J. Wright, “Initial algebra semantics and continuous algebras”, *Journal Assoc. Comp. Mach.*, vol. 24, 68 - 95, 1977.

[41] J.A. Goguen, R.M. Burstall, “Introducing Institutions”, in *Logics of Programs*, LNCS, vol. 164, Springer, 1983.

[42] J.A. Goguen, *What is unification? A categorical view of substitution, equation, and solution*, Tech. Rep. CSLI-88-124, Center for Study of Languages and Information, 1988.

[43] I. Guessarian, *Algebraic Semantics*, LNCS vol. 99, Springer, 1981.

[44] J.V. Guttag, *The Specification and Application to Programming of ADT*, Tech. Rep. CSR6-59, Computer Systems Research Group, University of Toronto, 1975.

[45] G.E. Hansoul, “A subdirect decomposition theorem for multialgebras”, *Algebra Universalis*, vol. 16, 275-281, 1983.

[46] D. Harel, A.R. Meyer, V.R. Pratt, “Computability and Completeness in Logics of Programs”, in Proceedings of *9th Annual ACM Symposium on Theory of Computing*, 1977.

[47] D. Harel, V.R. Pratt, “Nondeterminism in Logics of Programs”, in Proceedings of *5th Annual ACM Symposium on Principles of Programming Languages*, 1978.

[48] R. Heckmann, “Power Domains Supporting Recursion and Failure”, in *CAAP’92*, LNCS, vol. 581, Springer, 1992.

[49] M.C.B. Hennessy, “The semantics of call-by-value and call-by-name in a nondeterministic environment”, *SIAM J. Comput.*, vol. 9, no. 1, 1980.

[50] M. Hennessy, R. Milner, “On Observing Nondeterminism and Concurrency”, in *Automata, Languages and Programming*, LNCS, vol. 85, Springer, 1980.

[51] M. Hennessy, “Powerdomains and nondeterministic recursive functions”, in *Proc. 5th International Symp. on Programming*, LNCS, vol. 137, Springer, 1982.

[52] M. Hennessy, “Observing Processes”, in *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS, vol. 354, Springer, 1988.

[53] W.H. Hesselink, “A Mathematical Approach to Nondeterminism in Data Types”, *ACM: Transactions on Programming Languages and Systems*, vol. 10, 1988.

[54] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International Ltd., 1985.

[55] P.F. Hoogendik, “Relational programming laws in Boom hierarchy of types”, in Proceedings of *Mathematics of Program Construction*, Springer, 1992.

[56] G. Hornung, P. Raulefs, "Terminal algebra semantics and retractions for abstract data types", in *Automata, Languages and Programming*, LNCS, vol. 85, Springer, 1980.

[57] H. Hußmann, "Nondeterministic algebraic specifications and nonconfluent term rewriting", in *Algebraic and Logic Programming*, LNCS, vol. 343, Springer, 1988.

[58] H. Hußmann, *Nondeterministic Algebraic Specifications*, Ph.D. thesis, Fakultät für Mathematik und Informatik, Universität Passau, 1990.

[59] H. Hußmann, *Nondeterminism in Algebraic Specifications and Algebraic Programs*, Birkhäuser, 1993.

[60] G. Huet, J. Hullot, "Proofs by Induction in Equational Theories with Constructors", *JCSS*, vol. 25, 239-266, 1981.

[61] B. Jayaraman, "Implementation of Subset-Equational Programs", *Journal of Logic Programming*, vol. 12, no. 4, 299-324, 1992.

[62] S. Kaplan, "Conditional Rewriting", in *Conditional Term Rewriting Systems*, LNCS, vol. 308, Springer, 1987.

[63] S. Kaplan, "Rewriting with a Nondeterministic Choice Operator", *Theoretical Computer Science*, vol. 56, 37-57, 1988.

[64] D. Kapur, *Towards a theory of abstract data types*, Ph.D. thesis, Laboratory for CS, MIT, 1980.

[65] J.R. Kennaway, C.A.R. Hoare, "A theory of nondeterminism", in *Automata, Languages and Programming*, LNCS, vol. 85, Springer, 1980.

[66] P.R. Kosinski, "A Straightforward Denotational Semantics for Non-Determinate Data Flow Programs", *5th Annual Symposium on PoPL*, 1978.

[67] P.R. Kosinski, *Denotational Semantics of Determinate and Non-Determinate Data Flow Programs*, Ph.D. thesis, MIT Laboratory of Computer Science, 1979.

[68] D. Kozen, "Semantics of probabilistic programs", *J. Comput. Sys. Sci.*, vol. 22, 1981.

[69] V. Kriauciukas, M. Walicki, *Reasoning and Rewriting with Set-Relations I: Ground Case*, CSL'94 [also Tech. Rep. 92, University of Bergen, Dept. of Informatics, 1994.]

[70] R. Kuiper, *An operational semantics for bounded nondeterminism equivalent to a denotational one.*, Tech. Rep. IW 169/81, Stichtig Mathematisch Centrum, Dept. of CS, Amsterdam, 1981.

[71] D. Lehmann, S. Shelah, "Reasoning with Time and Chance", in *Automata, Languages and Programming*, LNCS, vol. 154, Springer, 1983.

[72] J. Levy, J. Augusti, "Bi-rewriting, a term rewriting technique for monotonic order relations", in *RTA'93*, LNCS, vol. 690, Springer Verlag, 1993, pp. 17-31.

[73] U. de Liguoro, A. Piperno, "Must pre-order in non-deterministic untyped lambda-calculus", in *CAAP'92*, LNCS, vol. 581, Springer, 1992.

[74] R.C. Lyndon, "Properties Preserved under Homomorphism", *Pacific Journal of Mathematics*, vol. 9, 1959.

[75] B. Möller, "On the Algebraic Specification of Infinite Objects - Ordered and Continuous Models of Algebraic Types", *Acta Informatica*, vol. 22, 537-578, 1985.

[76] B. Mahr, J.A. Makowsky, "Characterizing specification languages which admit initial semantics", in *Proc. 8th CAAP*, LNCS, vol. 159, Springer, 1983, pp. 300-316.

[77] T.S.E. Maibaum, *The Semantics of Nondeterminism*, Tech. Rep. CS-77-30, University of Waterloo, Ontario, Canada, December 1977.

[78] J.A. Makowsky, "Why Horn Formulas Matter in Computer Science", *Journal of Computer and System Science*, vol. 34, 266-292, 1987.

[79] A.I. Mal'cev, *The Metamathematics of Algebraic Systems*, Studies in Logic and the Foundations of Mathematics vol. 66, North-Holland Publishing Company, 1971.

[80] A.I. Mal'cev, *Algebraic Systems*, Die Grundlehren der mathematischen Wissenschaften ... Springer, 1973.

[81] A. Manes, "Fuzzy Theories", *Journal of Mathematical Analysis and Applications*, 1982.

[82] Z. Manna, "The Correctness of Nondeterministic Programs", *Artificial Intelligence*, vol. 1, 1 - 26, 1970.

[83] Z. Manna, A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer, 1992.

[84] J. McCarthy, "A basis for a mathematical theory of computation", in *Computer Programming and Formal Systems*, North-Holland, 1963.

[85] S. Meldal, "An Abstract Axiomatization of Pointer Types", in Proceedings of *The 22nd Annual Hawaii International Conference on System Sciences*, IEEE Computer Society Press, 1989.

[86] J. Meseguer, *A Logical Theory of Concurrent Objects*, Tech. Rep. SRI-CSL-90-70, SRI International, 1990.

[87] J. Meseguer, "Conditional rewriting logic as a unified model of concurrency", *TCS*, no. 96, 73-155, 1992.

[88] J. Mezei, J.B. Wright, "Algebraic automata and context-free sets", *Information and Control*, vol. 11, 1967.

[89] R. Milner, "Processes: a mathematical model of computing agents", *Proc. Logic Colloquium*, 1973.

[90] R. Milner, *Calculi for Communicating Systems*, LNCS vol. 92, Springer, 1980.

[91] P.D. Mosses, "Unified Algebras and Institutions", in Proceedings of *LICS'89, Fourth Annual Symposium on Logic in Computer Science*, 1989.

[92] P.D. Mosses, "Unified Algebras and Action Semantics", in *STACS'89*, LNCS, vol. 349, Springer, 1989.

[93] T. Nipkow, "Non-deterministic Data Types: Models and Implementations", *Acta Informatica*, vol. 22, 629 - 661, 1986.

[94] T. Nipkow, "Observing nondeterministic data types", in *Recent Trends in Data Type Specification*, LNCS, vol. 332, Springer, 1987.

[95] T. Nipkow, *Behavioural Implementations Concepts for Nondeterministic Data Types*, Ph.D. thesis, Dept. of CS, The University of Manchester, 1987.

[96] M. Nivat, "On the interpretation of recursive polyadic program schemes", *Symposia Mathematica*, vol. 15, 255-281 1975.

[97] M. Nivat, “Nondeterministic programs: an algebraic overview”, in *Information Processing ’80, Proc. of the IFJP Congress ’80*, North-Holland Publishing Company, 1980.

[98] M. O'Donnell, “Computing in Systems Described by Equations”, LNCS, vol. 58, Springer, 1977.

[99] D. Park, “On the Semantics of Fair Parallellism ”, in *Abstract Software Specifications*, LNCS, vol. 86, Springer, 1979.

[100] C.A. Petri, *Non-sequential processes*, Tech. Rep. ISF-77-05, Gesellschaft für Matematik und Datenverarbeitung, Sankt Augustin, 1977.

[101] G. Pickert, “Bemerkungen zum Homomorphie-Begriff”, *Mathematische Zeitschrift*, vol. 53, 1950.

[102] H.E. Pickett, “Homomorphisms and subalgebras of multialgebras”, *Pacific Journal of Mathematics*, vol. 21, 327-342, 1967.

[103] G. Plotkin, “A power domain construction”, *SIAM Jour. Comp.*, vol. 5, no. 3, 452 - 487, 1976.

[104] G. Plotkin, “Dijksrta’s Predicate Transformers and Smyth’s Powerdomains”, in *Abstract Software Specifications*, LNCS, vol. 86, Springer, 1980.

[105] G. Plotkin, K.R. Apt, “Countable Nondeterminism and Random Assignment”, *Tech. Rep. University of Edinburgh*, 1982.

[106] G. Plotkin, “Domains”, 1983, lecture notes, Dept. of Computer Science, University of Edinburgh.

[107] X. Qian, A. Goldberg, “Referential Opacity In Nondeterministic Data Refinement”, *ACM LoPLaS*, vol. 2, no. 1-4, 233-241, 1993.

[108] J.H. Reif, “Logics for probabilistic programming”, in Proceedings of *Twelfth ACM Symposium on Theory of Computing,* 1980.

[109] W. Reisig, *Petri Nets. An Introduction*, EATCS Monographs on Theoretical Computer Science vol. 4, Springer, 1985.

[110] H. Søndergaard, P. Sestoft, *Non-Determinacy and Its Semantics*, Tech. Rep. 86/12, Datalogisk Institut, Københavns Universitet, January 1987.

[111] N. Saheb-Djahromi, *Probabilistic CPO's for Nondeterminism*, Tech. Rep. CSR-37-79, University of Edinburgh, Dept. of CS, 1979.

[112] D. Sannella, A. Tarlecki, “Specifications in Arbitrary Institutions”, *Information and Computation*, vol. 76, 165-210, 1988.

[113] R.L. Schwartz, “An axiomatic treatment of ALGOL 68 routines”, in Proceedings of *Sixth Colloquium on Automata, Languages and Programming,* vol. 71, Springer, 1979.

[114] J. Schwartz, R. Dewar, E. Schonberg, E. Dubinsky, *Programming with Sets; An Introduction to SETL*, Springer, 1986.

[115] D. Scott, “Continuous Lattices”, in Proceedings of *1971 Dalhousie Conference,* LNM vol. 274, Springer, 1972.

[116] D. Scott, “Data Types as Lattices”, *SIAM Journal of computing*, vol. 5, no. 4, 522-587, 1976.

[117] M.B. Smyth, “Power domains”, *J. of Computer and System Sciences*, vol. 16, 1978.

[118] M.B. Smyth, “Power Domains and Predicate Transformers: A Topological View”, *Tech. Report, University of Edinburgh*, no. CSR-126-83, 1983.

[119] E.W. Stark, “Compositional relational semantics for indeterminate dataflow networks”, in *Category Theory and Computer Science*, LNCS, vol. 389, Springer, 1989.

[120] F. Stolzenburg, “An Algorithm for General Set Unification”, in Proceedings of *Workshop on Logic Programming with Sets, ICLP’93,* 1993.

[121] P.A. Subrahmanyam, “Nondeterminism in Abstract Data Types”, in *Automata, Languages and Programming,* LNCS, vol. 115, Springer, 1981.

[122] A. Tarlecki, “Free constructions in algebraic institutions”, in *Mathematical Foundation of Computer Science ‘84,* LNCS, vol. 176, Springer, 1984.

[123] E. Voermans, “Pers as types, inductive types and types with laws”, in Proceedings of *PHOENIX Seminar and Workshop on Declarative Programming,* Springer, 1991.

[124] H. Volger, *The Semantics of Disjunctive Deductive Databases*, Tech. Rep. MIP-8931, Fakultät für Mathematik und Informatik, Universität Passau, October 1989.

[125] H. Volger, “The semantics of disjunctive deductive databases”, in *CSL’89*, LNCS, vol. 440, Springer, 1989.

[126] M. Walicki, S. Meldal, “A Complete Calculus for the Multialgebraic and Functional Semantics of Nondeterminism”, 1993, [to appear in *ACM TOPLAS*].

[127] M. Walicki, *Algebraic Specifications of Nondeterminism*, Ph.D. thesis, University of Bergen, Department of Informatics, 1993.

[128] M. Walicki, S. Meldal, “Initiality+Nondeterminism => Junk”, *Proc. of NIK'93*, 1993.

[129] M. Walicki, S. Meldal, “Sets and Nondeterminism”, in Proceedings of *Workshop on Logic Programming with Sets: ICLP’93,* 1993.

[130] M. Walicki, S. Meldal, “Multialgebras, Power Algebras and Complete Calculi of Identities and Inclusions”, [to appear in *Recent Trends in Data Type Specifications,* LNCS 1995.]

[131] M. Walicki, *Singular and Plural Nondeterministic Parameters: Multialgebras, Power Algebras and Complete Reasoning Systems*, Tech. Rep. 96, Institutt for Informatikk, Universitetet i Bergen, 1994.

[132] M. Walicki, M. Broy, *Structured Specifications and Implementation of Nondeterministic Data Types*, Tech. Rep. Technische Universität München, Institut für Informatik, 1995.

[133] M. Walicki, S. Meldal, “ Generated Models and the Omega-rule; the Nondeterministic Case”, in Proceedings of *TAPSOFT'95,* 1995.

[134] G. Winskel, “Event structure semantics of CCS and related languages”, in *ICALP’82*, LNCS, vol. 140, Springer, 1982.

[135] G. Winskel, *Event structures*, LNCS vol. 255, Springer, 1987.

[136] G. Winskel, “An introduction to event structures”, LNCS, vol. 354, Springer, 1988.

[137] U. Wolter, M. Löwe, “Beyond Conditional Equations”, in *CAAP’92*, LNCS, vol. 581, Springer, 1992.