Multialgebras, Power Algebras and Complete Calculi of Identities and Inclusions*

Michal Walicki and Sigurd Meldal
Department of Informatics, University of Bergen
HiB, N–5020 Bergen, Norway
{michal,sigurd}@ii.uib.no

Abstract: After motivating the introduction of nondeterministic operators into algebraic specifications, a language $\mathcal{L}$ with two primitive predicates, *identity* and *inclusion*, for specifying nondeterministic operations is introduced. It is given a *multialgebraic* semantics which captures the *singular* (call-time-choice) strategy of passing nondeterministic parameters. A calculus NEQ, with restricted substitutivity rules, is introduced. NEQ is sound and complete wrt. the multialgebraic semantics.

A language $\mathcal{L}^*$ is obtained by a slight modification of $\mathcal{L}$ admitting *plural* (run-time-choice) parameters. The multialgebraic semantics is not sufficient for modeling such parameters and it is generalized to *power algebras*. Augmenting NEQ with one rule for unrestricted substitutivity for the plural variables yields NEQ* which is sound and complete wrt. to the power algebra semantics.

1. Introduction

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 descibed ("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 specificaiton.

We argue that 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.

Having established our motivation for using nondeterministic operators, we discuss the distinction between two modes of parameter passing – "call by value" and "call by name." In deterministic programming this distinction 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 parameters which are function expressions, passed by a kind of Algol copy rule [23], and which are evaluated whenever a need for their value arises. Thus call-by-name will terminate in many cases when the value of a function may be determined without looking at (some of) the actual parameters, i.e., even if these parameters are undefined. Call-by-value will, in such cases,

* This work has been partially supported by the Architectural Abstraction project under NFR (Norway), by CEC under ESPRIT–II Basic Reearch Working Group No. 6112 COMPASS, by the US DARPA under ONR contract N00014-92-J-1928, N00014-93-1-1335 and by the US Air Force Office of Scientific Research under Grant AFOSR-91-0354.

lead to an undefined result of the call. Nevertheless, the call-by-value semantics is usually preferred in the actual programming languages since it results in clearer and more tractable programs.

The nondeterministic counterparts of these two notions[1] are what we call *singular* (also called *call-time-choice* and corresponding to call-by-value) and *plural* (*run-time-choice* corresponding to call-by-name) parameter passing [2, 7, 24]. In the context where one allows nondeterministic parameters the difference between the two semantics becomes quite obvious even without looking at their termination properties. Let us suppose that we have defined an operation $g(x)$ as "***if*** x=0 ***then*** 0 ***elseif*** x=1 ***then*** 1 ***else*** 2", and that we have a nondeterministic choice operation "$\sqcup$._: Set(S)→ S" returning an arbitrary element from the argument set. The singular interpretation of $g(\sqcup.\{0,1\})$ will yield either 0 or 1, i.e., the result set of $g(\sqcup.\{0,1\})$ is {0,1}. The plural interpretation will give {0,1,2} as the set of possible results. (In a deterministic environment both semantics would yield the same results for this example.)

Another important difference concerns reasoning in the presence of nondeterministic operations, in particular, the substitutivity property. The inside-out substitution (corresponding, roughly speaking, to singular parameters) is not associative in the nondeterministic context [3, 4], and complicates the reasoning system by requiring specific restrictions on the substitution rules [12, 25]. Plural parameters, on the other hand, admit unrestricted substitution rules and, although semantically more complex, lead to simpler reasoning systems [25].

The above observations, together with the fact that the distinction has not received a thorough algebraic treatment,[2] motivate our investigation.

Multialgebras, used to model *singular* parameters, are algebras where operations are interpeted as set-valued functions and composition is defined by pointwise extension. This reflects the fact that, when the argument to an operation is a "set" (i.e., a nondeterministic expression), the choice of denotation for the expression (i.e. which element is to be used) is made at call-time, before passing the argument to the body of the operation. To model *plural* arguments, one has to generalize this construction and allow passing "whole sets" as arguments. This is achieved by using power algebras – algebras with carriers being (subsets of) power sets, and with operations mapping sets to sets (which in this setting are just the elements of the carriers).

In section 2 we give a general motivation for introducing nondeterministic operators as specification tools. In section 3 we define the language for specifying nondeterministic operators and its multialgebraic semantics which allow us to present, in section 4, two examples illustrating the usefulness of nondeterminism in achieving appropriate levels of abstraction. In section 5 we introduce a sound and complete calculus and discuss some of its features. Then we present an *algebraic* perspective on the distinction between the singular and plural passing of nondeterministic parameters. In section 6 the multialgebraic semantics for singular parameters is generalized to *power algebras* capable of modeling plural parameters. The corresponding sound and complete extension of the calculus is discussed in section 7. A comparison of both semantics in section 8 is guided by the similarity of the respective calculi. We indicate the

---

[1] We are not focusing here on the related distinctions (such as eager vs. lazy, IO vs. OI evaluation), discussion of which is beyond the scope of this paper.

[2] Unified algebras [19, 20] of P.D.Mosses, and rewriting logic [17, 16] of J. Meseguer handle both kinds of parameters. However, they do it in a highly non-standard, albeit elegant, way. We feel that multi- and power algebras stay closer to the traditional algebraic framework.

increased complexity of the power algebra semantics reflecting the problems with intuitive understanding of plural arguments. We also point out that plural variables can be used meaningfully to increase expressibility of the specification formalism even if all operations have only singular arguments.

The main (completeness) proofs are quite long and involved. The space limitation does not allow us to include them here, but all the proofs may be found in [26].

2. Nondeterministic Operators as Specification Tools

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.

(2) *Representational* nondeterminism.
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 dealt with uniformly within our framework, these 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 necessesarily accepting abstraction over *state* as requiring nondeterminsm for specification purposes.

How does this use of nondeterminism differ from the usual notion of underspecification? Consider for a moment a 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 $\sqcup$ 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 the head-function, 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 $\sqcup$ 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).

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 away 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 away from timing properties would introduce a seeming nondeterminism.

In order to make further examples more understandable, we have to introduce a formal language for specifying (possibly nondeterministic) operators and its semantics.

3. The Language $\mathcal{L}$ and the Multialgebra Semantics

A specification is a pair (Σ, Π), where the *signature* Σ is a pair of a sets (S, F) of sorts S and operation symbols F (with argument and result sorts in S). There exists a denumerable set V of variables for every sort. For any syntactic entity (term, formula, set of formulae) χ, $V[\chi]$ will denote the set of variables in χ.

The set of terms over the signature Σ and a variable set X is denoted $W_{\Sigma,X}$. We always assume that the set of ground terms of every sort S, S^{W_Σ}, is not empty.[1]

Π is a set of *sequents* of atomic formulae written as $a_1,\ldots,a_n \mapsto e_1,\ldots,e_m$. The left hand side of $\mapsto$ is called the *antecedent* and the right hand side the *consequent*, and both are to be understood as sets of atomic formulae (i.e., the ordering and multiplicity of the atomic formulae do not matter). In general, we allow either antecedent or consequent to be empty, in which case $\emptyset$ is usually dropped in the notation. A sequent with exactly one formula in the consequent (m=1) is called a *Horn formula*, and a Horn formula with empty antecedent (n=0) is a *simple formula* (or a *simple sequent*).

An atomic formula is either an *equation*, t=s, or an *inclusion*, $t \prec s$, of terms t,

[1] We do not address the problem of empty sorts here and will present calculi which work under the assumption that sorts are not empty. We use signatures with at least one constant for every sort but other ways of approaching this problem [5, 6, 11] seem to be compatible with our framework.

$s \in W_{\Sigma X}$. All variables occurring in a sequent are implicitly universally quantified over the whole sequent. For a specification SP=(Σ, Π), $\mathcal{L}$(SP) is the restriction of $\mathcal{L}$ to $W_{\Sigma,V}$.

The semantics of $\mathcal{L}$ specifications uses multistructures. Our definitions are very similar to those used by other authors [9, 12, 13, 21] except for the notion of equality. Also, we provide the means to interpret the occurrences of terms in $\mathcal{L}$ as *applications* of (possibly nondeterministic) operations rather than, as it is usually the case, as the sets of possible results.

Definition 3.1 (Multistructures). Let SP be an $\mathcal{L}$ specification. M is an SP-*multistructure* if

1. its carrier |M| is an S-sorted set and
2. for every $f: S_1 \times \ldots \times S_n \rightarrow S$ in F there is a corresponding function $f^M: S_1{}^M \times \ldots \times S_n{}^M \rightarrow \mathcal{P}^+(S^M)$.

□

where $\mathcal{P}^+$ denotes the power set with the empty set excluded. We let *MStr(SP)* denote the class of SP-multistructures. It has the distinguished term structure:

Definition 3.2 (Term multistructure). The *term multistructure* W_Σ for a specification SP=(Σ,Π) is defined as:

1. for each $S \in S$, S^{W_Σ} is the set of ground terms of sort S,
2. for each $f: S_1 \times \ldots \times S_n \rightarrow S$ in F, $t_i \in S_i{}^{W_\Sigma}$: $f^{W_\Sigma}(t_1 \ldots t_n) = \{f(t_1 \ldots t_n)\}$

□

It is a known fact that, in the general case, one cannot guarantee the existence of initial multimodels. Hußmann [12] has shown that even if we restrict $\mathcal{L}$ to simple formulae such multimodels may not exist. Therefore we admit general, and not only Horn, formulae in the specifications and will consider the whole class of multimodels of a specification.[1] The significance of the term multistructure is then summarized in

Lemma 3.3. If M is an SP-multistructure then for every set X of variables and assignment $\beta: X \rightarrow |M|$, there exists a unique function $\beta[\_]$: $W_{\Sigma X} \rightarrow \mathcal{P}^+(|M|)$ such that:

$\beta[x] = \{\beta(x)\}$, $\beta[c] = c^M$ and $\beta[f(t_1 \ldots t_n)] = \{f^M(t1 \ldots tn) \mid ti \in \beta[t_i]\}$

□

Application of multialgebraic operations to sets is defined by pointwise extension. Consequently, all operations in multistructures are $\subseteq$-monotonic, i.e., $\beta[s] \subseteq \beta[t] \Rightarrow \beta[f(s)] \subseteq \beta[f(t)]$.

Definition 3.4. An SP-multistructure M *satisfies* an $\mathcal{L}$(SP) sequent

$$\pi: t_i = s_i, \ldots, t_j \prec s_j \mapsto p_n = r_n, \ldots p_m \prec r_m,$$

written $M \models \pi$, iff for every assignment $\beta : X \rightarrow |M|$ we have

---

[1] For a discussion of initiality the reader is referred to [12, 25]. All the results reported in this paper remain valid for the specification language restricted to Horn formulae.

$$\beta[t_i]\equiv\beta[s_i] \wedge \ldots \wedge \beta[t_j]\subseteq\beta[s_j] \Rightarrow \beta[p_n]\equiv\beta[r_n] \vee \ldots \vee \beta[p_m]\subseteq\beta[r_m]$$

where $A\equiv B$ iff A and B are the same 1-element set.

An SP-*multimodel* is an SP-multistructure which satisfies all the axioms of SP. *MMod(SP)* denotes the class of multimodels of SP.

□

As a consequence of this definition, = is not an equivalence relation (it is not reflexive). $\mapsto t{=}t$ holds in a multialgebra M only if t^M has exactly one element, i.e., if the term t is deterministic. Of course, the set equality of two terms is expressible as two inclusions: $\mapsto s{\prec}t$ and $\mapsto t{\prec}s$.

Note that all variables are used singularly, i.e., they range over individuals (1-element sets) and not over arbitrary sets. In particular, assignments in lemma 3.3 and definition 3.4 assign to each variable a 1-element set. This fact is utilized to distinguish between the *result set* of an operation (which is represented by the corresponding term) and the result returned by a *particular application* of the operation as the following example illustrates.

Example 3.5.

The axiom $\mapsto x{\sqcup}y{=}x,\ x{\sqcup}y{=}y$ would make the binary choice operation $\_{\sqcup}\_$: $S{\times}S{\to}S$ deterministic (though underspecified). It says that (for any value of x,y), the set $x{\sqcup}y$ is either the same as the 1-element set x or y. In order to make $\sqcup$ a nondeterministic choice we have to say that *any application* of $x{\sqcup}y$ returns either x or y. This is expressed by the axiom: $z{\prec}x{\sqcup}y \mapsto z{=}x,\ z{=}y$.

□

4. Two Examples

Consider the problem of generating a depth-first traversal tree of nodes reachable from a particular node in a directed graph. The algorithm is found in standard algorithms textbooks (e.g. [15]), and is often given imperatively along the following lines (G is the graph, v is the start node, T is traversal tree being created and edges are ordered pairs of nodes):

Example 4.1.a

```
DFS(G,v) =
      begin T := ∅;
            trav(G,v,T);
            return T;
      end;
trav(G,v,T) =
      begin mark v;
            for all edges (v,x) do
              if x is unmarked then trav(G,x,T); T := T∪(v,x); endif ;
            endloop;
      end;
```

□

Now, consider an equational definition of DFS as a deterministic function. Let the function $n(\_,\_)$: Graph × Node → Set(Node) return the set of neighbors of a node in a given graph.

Example 4.1.b
$G, v, T, S, x \in V$:

$$\mapsto \mathrm{DFS}(G,v) = trav(G,v,\emptyset)$$

$$n(G,v)=\emptyset \mapsto trav(G,v,T) = T$$

$$\left.\begin{array}{l} n(G,v) = add(S,x), \\ x \in T = True \end{array}\right\} \mapsto trav(G,v,T) = trav(G\backslash\{(v,x)\}, v, T)$$

$$\left.\begin{array}{l} n(G,v) = add(S,x), \\ x \in T = False \end{array}\right\} \mapsto trav(G,v,T) = trav(G\backslash\{(v,x)\}, v, trav(G\backslash\{(v,x)\}, x, T\cup\{(v,x)\}))$$

□

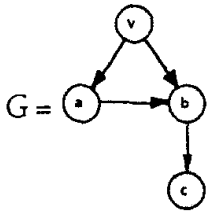
(The element tests check whether a node is in the tree (i.e., marked) already.) This definition looks plausible only as long as we do not inspect the Set sort. Adding elements to a set should be commutative – we have that

$$\mapsto add(add(S,x),y) = add(add(S,y),x)$$

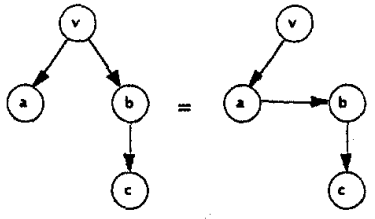
But then we also obtain

$$\mapsto trav(\ add(add(G,(v,a)),(v,b)),\ v,\ \emptyset) = trav(\ add(add(G,(v,b)),(v,a)),\ v,\ \emptyset)$$

In other words, for the graph



we get the DFS-tree equality:



which was not at all the intention – it collapses distinct tree values.

The problem is that the internal structure of the set value (in this case, the definition of DFS in terms of adding elements to the set) intrudes into the specification, quite contrary to the central tenet of abstract specifications.

An abstract definition of the DFS operator could be

Example 4.1.c
$G, v, T, S, x, y \in V$:

$$\mapsto \mathrm{DFS}(G,v) \prec trav(G,v,\emptyset)$$

$$n(G,v)=\emptyset \mapsto trav(G,v,T) = T$$

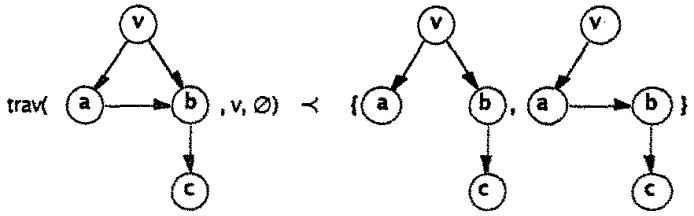
$$\left.\begin{array}{l} n(G,v) = add(S,y), \\ x \prec \sqcup.n(G,v), \\ x \in T = True \end{array}\right\} \mapsto trav(G,v,T) \prec trav(G\backslash\{(v,x)\}, v, T)$$

$$\left.\begin{array}{l} n(G,v) = add(S,y), \\ x \prec \sqcup.n(G,v), \\ x \in T = False \end{array}\right\} \mapsto trav(G,v,T) \prec trav(G\backslash\{(v,x)\}, v, trav(G\backslash\{(v,x)\}, x, T\cup\{(v,x)\}))$$

□

Though the specification still makes use of the structure of the set-generator functions, this no longer intrudes into the valuation of the function itself beyond ensuring a distinction between empty neighbor-sets and non-empty such. The variable x denotes the result of an arbitrary choice among these neighbors. The resultant definition defines the function without being concerned with (or specifying) the internal, repre-

sentational structure of the graph. We no longer collapse "distinct" trees, instead we only get the following (and plausible) result; that DFS-traversal will generate one of two trees:

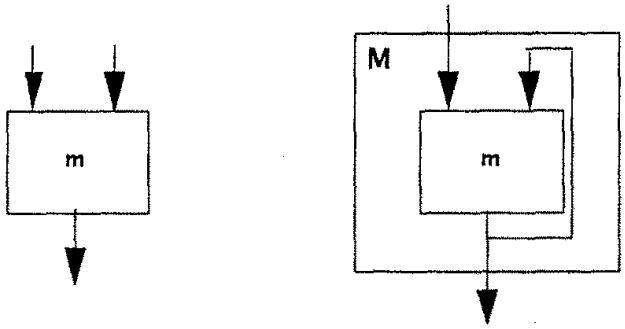


In general, iterating over sets or other structures is a natural operation, even when there is no intrinsic total order on the elements of such a structure. Such iteration is often deterministic, but representation-dependent, and if the iteration operation is *specified* as deterministic then we get an overspecification. The possibility to ignore the representation-dependent structure of the specified data is one of the fundamental requirements for a specification formalism.

The last small example illustrates another aspect of the abstraction potential inherent in nondeterminism.

Abstracting over *time* is such a natural thing to do that many consider timing dependencies as representing *real* nondeterminism. However, time is just another component of state, and abstraction over time could therefore be handled similarly to abstraction over state in general. As an example of how time can be removed from consideration in a specification, consider the specification of a merge function m. Its arguments are two streams of data, and the result a new stream which is a merger of the two (here the metric aspects of time have been removed, leaving time only in a vestigial form as a total order for each of the input- and output-streams, ignoring even the relative ordering of elements in distinct streams) (e.g. in [10, 18] and related works).

Let M be a function with only one input stream, but constructed from m with the output stream fed back as one of the input streams of m (see figure).



A specification of these two functions could be (where ^ is the concatenation operator on streams and ε is an empty stream):

Example 4.2

$q, p, x, y, r \in V$:

$$\begin{aligned} & \mapsto m(\varepsilon,q) = q \\ & \mapsto m(q,\varepsilon) = q \\ r \prec m(x\hat{}q, y\hat{}p) \; & \mapsto r \prec x\hat{}m(q, y\hat{}p),\; r \prec y\hat{}m(x\hat{}q, p) \\ & \mapsto M(\varepsilon) = \varepsilon \\ r \prec M(x\hat{}q) \; & \mapsto r \prec x\hat{}m(q, r) \end{aligned}$$

□

As we can see, m is deterministic when there is only one non-empty input stream. Buf if there are two, then the first element of the result stream (r) is the first element of *one or the other* of the two input streams. When we construct the feedback function M then any nonempty input stream results in an infinite output stream, but the composition of the output is not determined – that would reflect the timing property of the function evaluation and of the input, which we have abstracted. Again, the abstraction shows up as nondeterminism.

It may be an interesting excercise for the reader to convince himself that the above specification yields the intended meaning for the M operator and does not lead to the classical merge-anomalies [11, 14]. The example is discussed in more detail in [25].

Finally, we can mention that the close relation between nondeterministic terms and sets makes it possible to use the former to define and handle subsorting directly at the term level. This is the basic intuition behind the framework of unified algebras [19, 20] and can also be done within the formalism we have introduced here.

5. The Calculus NEQ

The last axiom in the merge example 4.2 uses the variable r to specify the desired properties of *each possible* result produced by the M operator. It says that any such r is obtained by taking the first element x of the input sequence and then merging the rest of the input sequence with r *itself*. This is different from the axiom obtained by replacing the occurrences of r with $M(x\hat{}q)$: $\mapsto M(x\hat{}q) \prec x\hat{}m(q,M(x\hat{}q))$ which, plausible as it may seem, creates the usual merge-anomalies.

This illustrates the inherent problem of reasoning with nondeterminism, namely, unsoundness of unrestricted substitution of terms for variables. In our formalism we handle this problem by turning = into a partial equivalence realtion and allowing substitution of a term t for a variable only if $\mapsto t{=}t$ is derivable (rule SB). The rules of the calculus NEQ are:

R1: $\quad \mapsto x{=}x \qquad x \in V$

R2: $$\frac{\Gamma_t^x \mapsto \Delta_t^x \quad ; \quad \Gamma' \mapsto s{=}t, \Delta'}{\Gamma_s^x, \Gamma' \mapsto \Delta_s^x, \Delta'}$$

R3: $$\frac{\Gamma \mapsto \Delta_t^x \quad ; \quad \Gamma' \mapsto s \prec t, \Delta'}{\Gamma, \Gamma' \mapsto \Delta_s^x, \Delta'} \qquad \text{x not in a RHS[1] of } \prec$$

R4: $\quad e \mapsto e$

R5: $$\frac{\Gamma \mapsto \Delta, e \quad ; \quad \Gamma', e \mapsto \Delta'}{\Gamma, \Gamma' \mapsto \Delta, \Delta'} \qquad \text{(CUT)}$$

R6: a) $$\frac{\Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, e}$$ b) $$\frac{\Gamma \mapsto \Delta}{\Gamma, e \mapsto \Delta}$$ (WEAK)

[1] RHS, resp. LHS, stand for right, resp. left, hand side.

R7: a) $\dfrac{\Gamma, x \prec t \mapsto \Delta}{\Gamma_t^x \mapsto \Delta_t^x}$ $x \in V \backslash V[t]$; at most one x in $\Gamma \mapsto \Delta$; x in Γ isn't in RHS of $\prec$

b) $\dfrac{\Gamma, x \prec t, y \prec r \mapsto \Delta}{\Gamma, y \prec r_t^x \mapsto \Delta}$ $x \in V \backslash V[t]$, $y \in V$; no x in $\Gamma \mapsto \Delta$; at most one x in r; x and y are distinct variables (ELIM)

χ_b^a denotes χ with b substituted for a. The rule R1 expresses the fact that only variables are guaranteed to be deterministic. The restriction on R3 prevents one, for instance, from drawing the unsound conclusion $s \prec p$ from the premises $s \prec t$ and $p \prec t$.

The R7 rules allow one to eliminate a redundant binding $x \prec t$ replacing x by t. Since x refers to one and the same value, such a replacement requires that there be at most one occurrence of x. Otherwise we could, for instance, derive $t=t$ (for *arbitrary* t) from $x \prec t \mapsto x = x$ by a single application of R7a.

A similar problem would occur if we removed the second or third restriction from R7b. The other restrictions are of purely technical character. Allowing x to occur in t would, for instance, lead to the unsound deduction (using R7a): $\dfrac{x \prec t(x), x = 1}{t(x) = 1}$

The last restriction in R7a excludes the special case which is related to the singular semantics and is treated by R7b.

As an example of the semantic import of thes rules we give a few derived rules:

NE: $\dfrac{x \prec t \mapsto}{\mapsto}$ Restrictions as for R7a (Terms represent non-empty sets)

SI: $\dfrac{x \prec t, s \prec r \mapsto \;;\; \mapsto s = s}{s \prec r_t^x \mapsto}$ Restrictions as for R7b (Arguments are singular)

This rule can be rephrased as $\dfrac{\forall x (x \in t \Rightarrow s \notin r(x))}{s \notin r(t)}$ which, read contravariantly, says: if s is in the result set of $r(t)$ then there exists an individual $x \in t$ such that s is in the result set $r(x)$, i.e., the argument t to r is singular.

SB: $\dfrac{\Gamma \mapsto \Delta \qquad \Gamma' \mapsto t=t, \Delta'}{\Gamma_t^x, \Gamma' \mapsto \Delta_t^x, \Delta'}$ (Variables are replacable by deterministic terms)

The main theorem states soundness and completeness of NEQ wrt. the multialgebraic semantics:

Theorem 5.1. For every specification SP=(Σ,Π), sequent $\pi \in \mathcal{L}$(SP)
$$\mathrm{MMod}(SP) \models \pi \text{ iff } \Pi \vdash_{NEQ} \pi$$

□

6. Power Algebras

Singular arguments have the usual algebraic property that they refer to a unique value. This corresponds to evaluation at the moment of substitution and passing the result to the following computation. Plural arguments, on the other hand, are best understood as textual parameters. They are not passed as a single value but the expression receiving them is first expanded and evaluation takes place only when the expansion is completed.

To capture this possibility we have to extend the multialgebra semantics to power algebras where operations map sets to sets. We will allow both singular and plural parameter passing in anyone specification. The corresponding semantic distinction will be between the power set functions which are merely $\subseteq$-monotonic and those which also are $\cup$-additive.

In the language we merely introduce a notational device for distinguishing the singular and plural arguments. We allow annotating the sorts in the profiles of the operation by a superscript, as in S^+, to indicate that an argument is plural. Furthermore, we partition the set of variables into two disjoint subsets of singular, X, and plural, X^+, variables. x and x^+ are to be understood as distinct symbols. We will say that an operation f is *singular in the i^{th} argument* iff the i^{th} argument (in its signature) is singular. The specification language extended with such annotations of the signatures will be referred to as $\mathcal{L}^+$. By a *singular specification* we will mean one where all arguments to all operations are singular.

Definition 6.1. Let each S_i be the power set of some underlying set $\underline{S}_i$, i.e., $S_i = \mathcal{P}^+(\underline{S}_i)$. A function f: $S_1 \times ... \times S_n \rightarrow S$ is $\cup$-*additive* in the i^{th} argument iff for all $x_i \in S_i$ and all $x_k \in S_k$ (for $k \neq i$) : $f(x_1 ... x_i ... x_n) = \bigcup \{ f(x_1 ... \{x\} ... x_n) \mid x \in x_i \}$.
⊡

Definition 6.2. Let Σ be a $\mathcal{L}^+$-signature. A is a Σ-*powerstructure*, $A \in PStr(\Sigma)$, iff A is a (deterministic) structure such that:

1. for every $S \in \mathbf{S}$, the carrier $S^A \subseteq \mathcal{P}^+(\underline{S})$, for some underlying set $\underline{S}$,
2. for every f: $S_1 \times ... \times S_n \rightarrow S$ in Σ, f^A is a $\subseteq$-monotonic function ${S_1}^A \times ... \times {S_n}^A \rightarrow S^A$ such that, if the i^{th} argument of f is singular then f^A is additive in the i^{th} argument.

⊡

Note the unorthodox point in definition 6.2 – we do not require the carrier of a power structure to be the whole power set but allow it to be a *subset* of some power set. All finite subsets are needed, for instance, if one assumes a primitive nondeterministic choice with predefined semantics of set union. We do not assume anything of the kind and expect that quite many meaningful specifications may do very well without all possible subsets. In addition, using full power sets as carriers would always yield unreachable structures whenever the underlying set is infinite.

Given an f:$S \times S^+ \rightarrow T$, we will say that an actual argument at the first position has a *singular occurrence*. E.g., in $f(t,t)$, the first t has a singular, and the second one a plural occurrence. More precisely:

Definition 6.3. α has a *singular occurrence* in a term t iff one of the following holds ($\doteq$ denotes syntactical identity):

1. $t \doteq \alpha$
2. $t \doteq f(...,\alpha,...)$ and f is singular in the argument corresponding to α,
3. $t \doteq f(t_1,...,t_n)$ and α has a singular occurence in one of t_i.

⊡

The first point is included for the technical reasons – the definition will be used to specify additional restrictions on the application of some reasoning rules. To define satisfiability of formulae by a power structure we only need to extend the definition of an assignment

Definition 6.4. Let X be a set of singular and X^+ a set of plural variables. By an *assignment* into a power structure A we mean a function $\beta: X \cup X^+ \to |A|$ such that, for all $x \in X$, $|\beta(x)| = 1$.
□

If β is as in this definition, then every term $t(x,x^+) \in W_{\Sigma,X,X^+}$ has a unique set interpretation $\beta[t(x,x^+)]$ in A defined as $t^A(\beta(x),\beta(x^+))$. Satisfiability of sequents over $\mathcal{L}^+(\Sigma,X,X^+)$ by a power structure is then defined exactly as before (def. 3.4) and the class of power models of a specification SP is denoted PMod(SP).

Since functions from A to $\mathcal{P}^+(B)$ are isomorphic to $\cup$-additive functions from $\mathcal{P}^+(A)$ to $\mathcal{P}^+(B)$, $[A \to \mathcal{P}^+(B)] \simeq [\mathcal{P}^+(A) \to_\cup \mathcal{P}^+(B)]$, we may consider every multistructure A to be a power structure A^+ by taking $|A^+| = \mathcal{P}^+(A)$ and extending all operations in A pointwise. We then have the obvious

Lemma 6.5. Let SP be a singular specification, $A \in$ MStr(SP), and π be a sequent in $\mathcal{L}$(SP). Then $A \models \pi$ iff $A \,^+\!\models \pi$, and so $A \in$ MMod(SP) iff $A^+ \in$ PMod(SP).
□

7. The Calculus NEQ$^+$ for Join Semantics

We let $V[t]$ be the set of singular and $V^+[t]$ the set of plural variables in t. Rules R1-R7 are as in NEQ (except for a new restriction in R7) but now all terms t_i belong to W_{Σ,X,X^+}. In particular, any t_i may be a plural variable.

R1: $\quad \mapsto x=x \qquad\qquad x \in V$

R2: $$\frac{\Gamma_t^x \mapsto \Delta_t^x \quad ; \quad \Gamma' \mapsto s=t, \Delta'}{\Gamma_s^x, \Gamma' \mapsto \Delta_s^x, \Delta'}$$

R3: $$\frac{\Gamma \mapsto \Delta_t^x \quad ; \quad \Gamma' \mapsto s \prec t, \Delta'}{\Gamma, \Gamma' \mapsto \Delta_s^x, \Delta'} \qquad x \text{ not in a RHS of } \prec$$

R4: $e \mapsto e$

R5: $$\frac{\Gamma \mapsto \Delta, e \quad ; \quad \Gamma', e \mapsto \Delta'}{\Gamma, \Gamma' \mapsto \Delta, \Delta'} \qquad \text{(CUT)}$$

R6: a) $\dfrac{\Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, e}$ $\qquad$ b) $\dfrac{\Gamma \mapsto \Delta}{\Gamma, e \mapsto \Delta}$ $\qquad$ (WEAK)

R7: a) $\dfrac{\Gamma, x \prec t \mapsto \Delta}{\Gamma_t^x \mapsto \Delta_t^x}$ $x \in V \setminus V[t]$; at most one x in $\Gamma \mapsto \Delta$; x in Γ isn't in RHS of $\prec$; the occurrence of x is *singular*

b) $\dfrac{\Gamma, x \prec t, y \prec r \mapsto \Delta}{\Gamma, y \prec r_t^x \mapsto \Delta}$ $x \in V \setminus V[t]$, $y \in V$; no x in $\Gamma \mapsto \Delta$; at most one x in r; the occurrence of x in r is *singular*; x and y are distinct variables (ELIM)

R8: $\dfrac{\Gamma \mapsto \Delta}{\Gamma_t^{x^+} \mapsto \Delta_t^{x^+}}$ (SUBP)

We used R7b from NEQ to derive SI, which expressed singularity of all arguments. Therefore, in NEQ$^+$ we need an additional restriction to make sure that the substitution for x takes place only at the arguments which are singular. The derived rules MO, NE, SB are the same as for NEQ, but SI is now restricted to the singular occurrences of x.

The new rule R8 expresses the semantics of plural variables. It allows us to substitute an arbitrary term t for a plural variable x^+. Taking t to be a singular variable x, we can thus exchange plural variables in a provable sequent π with singular ones. The opposite is, in general, not possible because rule R1 applies only to singular variables. Thus a plural variable x^+ will satisfy $\mapsto x^+ \prec x^+$, but this is not sufficient for performing a substitution for a singular variable according to SB.

The result corresponding to theorem 5.1 is:

Theorem 7.1. For any $\mathcal{L}^+$-specification SP and $\mathcal{L}^+$(SP) sequent π:

$$\mathrm{PMod(SP)} \models \pi \text{ iff } \Pi \vdash_{\mathrm{NEQ}^+} \pi$$

□

8. Singular vs. Plural, Arguments vs. Variables

NEQ$^+$ has the additional rule R8 which could suggest that more formulae are derivable with it than with NEQ. This would go counter lemma 6.5 and the intuition that power models form a more general class than multimodels. There is no contradiction, however, because what actually limits the number of derivations in NEQ$^+$ is the additional restriction on the rules R7. For instance, having operations g:S→ T, and f:S$^+$ →T, we may in both calculi prove:

$$\frac{x \prec t, y \prec g(x) \mapsto}{y \prec g(t) \mapsto} \text{ R7b}$$

Replacing g with f in the assumption would disallow the analogous conclusion in NEQ$^+$.

Rule R8, admitting instantiation of plural variables, is useful only if the axioms of the specification contain such variables. Axioms with plural variables can also be viewed as axiom schemata for axioms without such variables. From the logical point of view, axiom $\mapsto f(x^+) \prec r(x^+, x^+)$ leads to the same formulae (without plural variables) as the set of axioms $\{ \mapsto f(t) \prec r(t,t) \mid t \in W_{\Sigma,X} \}$.

Thus we can see that rule R7$^+$ is concerned with plural *arguments*, while rule R8 with plural *variables*. In fact, introducing plural arguments does not force one to use plural variables and, on the other hand, axioms containing plural variables can be used even if all operations are singular. We may set up the relations between the use of singular/plural variables/arguments and the associated sound and complete rea-

soning systems in the following way ($R7^+$ denotes R7 with the restrictions from NEQ^+):

arguments / variables	singular	plural
singular	1 NEQ	2 NEQ,$R7^+$
plural	3 NEQ,R8	4 NEQ^+

If a specification contains only singular variables, then NEQ is sufficient for proving all its consequences if all operations are singular (1) – if some arguments are plural (2) then we have to use the more restricted version $R7^+$. Obviously, we have that $2 \subseteq 1$ and $4 \subseteq 3$.

If all operations are singular then we may still use plural variables in the formulae and need to extend NEQ with the rule R8 (3). In this case, we have to consider multialgebras as power algebras with all operations being additive (according to lemma 6.5), in order to obtain a proper notion of assignment to the plural variables. In fact, this is the alternative we would prefer in general, unless one is explicitly interested in the specification of plural arguments. We feel that this combination of the singular semantics of parameter passing with the use of plural variables gives us both simplicity of multialgebras (as compared to power algebras) and the increased expressive power in writing specification as illustrated by the following example.

Example 8.1.

Consider the following (singular) specification of binary choice $\_\sqcup\_$. $S \times S \to S$ as the join operator:

$$\mapsto x^+ \prec x^+ \sqcup y^+$$
$$\mapsto y^+ \prec x^+ \sqcup y^+$$
$$x^+ \prec z^+, y^+ \prec z^+ \mapsto x^+ \sqcup y^+ \prec z^+$$

An analogous attempt to specify join with singular variables only would fail, because the last axiom would then be $x \prec z, y \prec z \mapsto x \sqcup y \prec z$ which is equivalent to $\mapsto z \sqcup z = z$. This observation indicates that plural variables may be an alternative to disjunctions which had to be used for the specification of choice in example 3.5.

□

9. Conclusions and Further Work.

We have introduced a formalism for specification of (possibly) nondeterministic operations and defined multialgebra and power algebra semantics for the singular, respectively, plural parameters. The main result of the paper are the two reasoning systems which are sound and complete for the respective semantics.

The comparison of the two semantics led us to point out that the singular/plural distinction has two, relatively independent, facets. On the one hand, it may be taken as a purely semantic distinction concerning the mechanism of parameter passing. On the other hand, plural variables may be used as a merely syntactic device to increase expressiveness of the specification language, which does not force one to accept the plural semantics of parameter passing.

We have considered only flat specifications and consequently the current results must be seen only as the first step toward a full specification formalism which would be applicable in software develompment practice. The work on structural specification with nondeterminism is in progress and we can only indicate some main points. The central idea is the one emphasized in this and other papers [7, 22, 25, 27]: *nondeterminism is a natural abstraction tool* and this fact may prove valuable when considering the implementation and composition of specifications.

Specification-building operations such as **enrich** (+), **derive**, (**reduct**) and hence **export** and **rename** should extend smoothly to the nondeterministic context.

Quotient needs a slight generalisation since we have only partial equivalence and not congruence. Releasing the congruency claim w.r.t. nondeterminisitc operations may seem a blasphemy to the mathematical practice, but it turns out to be a crucial move in achieving a sound data refinement in a nondeterminisitic setting. Our current experiences and [22] show that some problematic cases may be elegantly handled using our nondeterministic framework. Consider for instance the implementation of abstract sets with a (non- or underdetermined) choice operator. A natural and simple implementation would represent sets as sequences with the "head" operation implementing choice. Accepting this as a correct implementaiton would traditionally require the notion of *behavioral equivalence*. In such cases, the abstract character of nondeterministic operations may be used successfully as an alternative to the behavioural abstraction. Whether this is a viable way for a wider range of applications and whether this will allow one to limit the need for behavioral abstraction remains to be seen.

As we have observed, initial multialgebras do not exist even in very elementary cases. Since initiality and quotient are special cases of free extensions, one shouldn't expect much of the **extend-freely** operation. Reachable extensions seem still possible but one will face several choices of the notion of reachability [25].

References

1. Brock, J.D., Ackermann, W.B., "Scenarios: A model of non-determinate computation", in *Formalization of Programming Concepts*, LNCS, vol. 107, Springer, 1981.
2. Clinger, W., "Nondeterministic call by need is neither lazy nor by name", *Proc. ACM Symp. LISP and Functional Programming*, 226-234, 1982.
3. Engelfriet, J., Schmidt, E.M., "IO and OI. 1", *Journal of Computer and System Sciences*, vol. 15, 328-353, 1977
4. Engelfriet, J., Schmidt, E.M., "IO and OI. 2", *Journal of Computer and System Sciences*, vol. 16, 67-99, 1978.
5. Goguen, J.A., Meseguer, J., "Completeness of Many-Sorted Equational Logic", *SIGPLAN Notices*, vol. 16, no. 7, 1981.
6. Goguen, J.A., Meseguer, J., "Remarks on Remarks on Many-Sorted Equational Logic", *SIGPLAN Notices*, vol. 22, no. 4, 41-48, April 1987.
7. Hayes, I., Jones, C., "Specifications are not (necessarily) executable", in *Software Engineering Journal*, 4(6): 330-338. 1989.
8. Hennessy, M.C.B., "The semantics of call-by-value and call-by-name in a nondeterministic environment", *SIAM J. Comput.*, vol. 9, no. 1, 1980.
9. Hesselink, W.H., "A Mathematical Approach to Nondeterminism in Data Types", *ACM: Transactions on Programming Languages and Systems*, vol. 10, 1988.

10. Hoare, C.A.R., *Communicating Sequential Processes*, Prentice-Hall International Ltd., 1985.
11. Huet, G., Oppen, D., "Equations and Rewrite Rules: A Survey", in *Formal Language Theory: Perspectives and Open Problems*, Academic Press, 1980.
12. Hußmann, H., *Nondeterminism in Algebraic Specifications and Algebraic Programs*, Birkhäuser, 1993.
13. Kapur, D., *Towards a theory of abstract data types*, Ph.D. thesis, Laboratory for CS, MIT, 1980.
14. Keller, R.M., "Denotational models for parallel programs with indeterminate operators", in *Formal Descriptions of Programming Concepts*, North-Holland, Amsterdam, 1978.
15. Manber, U., *Introduction to Algorithms*, Addison-Wesley, 1989.
16. Meseguer, J., "Conditional rewriting logic as a unified model of concurrency", *TCS*, no. 96, 73-155, 1992.
17. Meseguer, J., "Conditional Rewriting Logic: Deduction, Models and Concurrency", in Proceedings of *CTRS'90*, LNCS vol. 516, 1990.
18. Milner, R., *Communication and Concurrency*, Prentice Hall International, 1989.
19. Mosses, P.D., "Unified Algebras and Action Semantics", in *STACS'89*, LNCS, vol. 349, Springer, 1989.
20. Mosses, P.D., "Unified Algebras and Institutions", in *Proceedings of LICS'89, Fourth Annual Symposium on Logic in Computer Science*, 1989.
21. Nipkow, T., "Non-deterministic Data Types: Models and Implementations", *Acta Informatica*, vol. 22, 629–661, 1986.
22. Qian, X., Goldberg, A., "Referential Opacity in Nondeterministic Data Refinement", in *ACM LoPLAS*, vol.2, no.1-4, 1993.
23. Schwartz, R.L., "An axiomatic treatment of ALGOL 68 routines", in Proceedings of *Sixth Colloquium on Automata, Languages and Programming*, vol. 71, Springer, 1979.
24. Søndergaard, H., Sestoft, P., *Non-Determinacy and Its Semantics*, Tech. Rep. 86/12, Datalogisk Institut, Københavns Universitet, January 1987.
25. Walicki, M., *Algebraic Specifications of Nondeterminism*, Ph.D. thesis, University of Bergen, Department of Informatics, 1993.
26. Walicki, M., *Singular and Plural Nondeterministic Parameters: Multialgebras, Power Algebras and Complete Reasoning Systems*, Tech. Rep. 96, Department of Informatics, University of Bergen, 1994.
27. Ward, N., *"A Refinement Calculus for Nondeterminisitc Expressions"*, PhD Thesis, Dept.. of Computer Science, The University of Queensland, submitted February 1994.