Structured Specifications and Implementation of Nondeterministic Data Types

Michał Walicki
Universitetet i Bergen
Institutt for Informatikk
HiB, 5020 Bergen, Norway
michal@ii.uib.no

Manfred Broy
Technische Universität München
Institut für Informatik, Arcisstr.21
80290 München, Germany
broy@informatik.tu-muenchen.de

Abstract

The use of nondeterminism in specifications, as distinct from underspecification, is motivated by an example in the context of data refinement. A simple formalism for specifying nondeterministic data types is introduced. Its semantics is given in terms of the existing formalisms of relations, multialgebras, sets of functions and oracles by means of appropriate translation rules. Nondeterministic data refinement is studied from the syntactic and semantic perspective, and the correctness of the suggested proof obligations is proved. More general, the implementation relation and parameterisation of nondeterministic data types are discussed and the standard theorems of vertical and horizontal composition are generalized to the nondeterministic case.

1 Introduction

Mathematicians never saw a reason to deal with something called "nondeterminism". In mathematics we work with values, functions, sets, relations. In computing science nondeterminism has been an issue from the beginning. It is often suggestive to consider models of computation where for certain operations[1] we leave some freedom of choice. We then speak about *sets of possible outcomes* rather than just *the value* returned by such an operation. Clearly, we then have to distinguish carefully between a deterministic program which computes a set of values, and a nondeterministic one which computes a single data element out of a set of possible values every time it is executed.

Many reasons for considering choices in computations may be gathered under the common name *"abstraction"*. In terms of modeling, nondeterminism may be considered a purely operational notion. But in the context of specification, which interests us here, appropriate abstraction is the central issue. Typically, we prefer to work with models which do not include all details of the physical environment of computation such as timing, temperature, representation by hardware etc. Since we do not want to model all these complex dependencies, we rather map such aspects by nondeterministic choices.

Furthermore, when developing a software system in a number of refinement steps, we are often interested in specifying the functionality of the system not uniquely but only with respect to some required properties. We then speak of *underspecification*. Later in the development process we may add more properties, whenever we find this appropriate. A conceptual difference between underspecification and nondeterminism does not contradict the fact that both arise as natural abstraction mechanisms and, sometimes, may be supported by similar modeling techniques.

In an axiomatic specification setting, the reasons for distinguishing the two are not only conceptual but, as the paper will illustrate, also technical. Roughly speaking, the difference consists

[1] We reserve the word *function* for deterministic operations. *Operation* may refer both to a function and to a nondeterministic operator.

in that underspecification corresponds to choices between models while nondeterminism to choices within one model. Working with the concept of loose specifications, we associate with a given specification a class of models. If some of these models are not isomorphic, we speak also of underspecification. Nondeterminism, on the other hand, is represented within each model by specific constructs such as relations, set-valued functions, sets of functions. This explicit introduction of nondeterminism has also its technical price. While underspecification fits into the concepts of classical logic and model theory smoothly, the treatment of nondeterminism leads to complications. Classical concepts of logical calculi such as substitutivity might no longer be valid. This was treated in full detail in [WM95a], [WM95b], and will be taken into account in the following.

Of course, we may (and in this paper we do) work with a specification method that supports both nondeterminism and underspecification. Since, on the one hand, underspecification is a simple, commonly used abstraction mechanism in deterministic context and, on the other hand, nondeterminism helps to handle some classical problems, we do not see any reasons to choose between them.

The refinement concepts for underspecification and for nondeterminism are rather similar. We work with loose specifications and a refinement step amounts to resolving underspecification by adding more axioms and yielding a more restricted class of models. Analogously, nondeterminism may be removed by restricting the set of possible outcomes of a nondeterministic operation. In fact, this latter restriction (of nondeterminism of some operations) is just a special case of the former one. It amounts to selecting, by means of additional axioms restricting nondeterminism of the operations, only these models which satisfy the extended set of axioms.

1.1 Algebraic Specification of Nondeterminism

In the algebraic framework, nondeterminism has been treated by several authors who introduced new kinds of non-standard models, such as rewriting logic [Mes92], or unified algebras [Mos89]. Their common feature is that nondeterministic operations do not, in general, correspond to *sets of possible results* but are modeled by means of special new constructs. One of the main motivations for these approaches is the interest in the initial semantics and the above mentioned approaches guarantee the existence of initial models.

We are interested in preserving the intuition of a nondeterministic operation as returning an element from some set of possible outcomes. In this tradition, one may distinguish two further branches. On the one hand, there are attempts to equate nondeterminism and underspecification, typically, by modeling the former as a collection of deterministic entities. Here, we can mention

- oracles
- sets of functions.

The other approach does not involve the implicit assumption that nondeterminism is just a lack of information about a computational entity which, at a sufficiently detailed level of description, is deterministic. It chooses model structures which may be used both when nondeterminism is eventually resolved into deterministic components and, on the other hand, when it is treated as a phenomenon *per se*, not necessarily reducible to determinism. Here we have, for instance

- multialgebras
- relations.

We will analyze and compare all these four possibilities and present an approach allowing the specifier to apply his favorite formalism for nondeterminism.

We aim at a minimal extension of the standard (equational) specifications allowing us to specify *possibly* nondeterministic operations. We say "possibly" because we will not see any examples where nondeterminism is required by the specification. Saying "nondeterministic" we mean "possibly nondeterministic" (and possibly not). Although we will introduce a formalism where one can require an operation to be nondeterministic, our general assumption is that nondeterminism is primarily an *abstraction concept*. It is used to describe the operations at a given level of abstraction where the possible parameters determining the exact (and ultimately, perhaps, deterministic) behavior are unknown or irrelevant. In this sense, it is similar to underspecification but, as it will turn out, requires a different treatment. Just as we do not assume that an operation, treated at the given level of abstraction as nondeterministic, is actually so, we neither assume that it is deterministic.

Furthermore, we consider it to be a realistic scenario that nondeterminism occurs only locally in a specification. The main part of the specification consists of the classical functions and only some (minor) parts are described as if they were nondeterministic. In particular, we think of a program as working on a deterministic basis – the set of function symbols is always assumed to be non-empty and we do not allow nondeterministic constructors.

1.2 Implementation

We study the development of nondeterministic data types by stepwise refinement and, as remarked above, look at nondeterminism primarily as an abstraction tool. For these reasons, we will focus on loose semantics.

However, the methodological postulate "nondeterminism is abstraction" does not necessarily imply that we never intend to reach a nondeterministic implementation. If a specification indicates an operation to be nondeterministic – how can we implement it? Say, we have two axioms, $a \prec c$ and $b \prec c$, requiring that both a and b are among the possible results returned by different invocations of c. In order to implement this, c must "remember" what values it has returned on previous calls to ensure that it does not forever ignore one of them. This seems to be the main issue of implementing nondeterminism – it requires that one considers not only what happens in a single application of an operation but, potentially, what happens in an arbitrarily long series of such applications.

One might consider a restricted version of a specification formalism which allows one to specify only a single application of an operation but not, like in the above axioms, a set of such applications. On the other hand, one might restrict the formalism so that nondeterminism in the implementation would be admissible but could not be required by the specification.

However, there are situations when one would like to ensure some liveness properties which would be expressed by a requirement of a certain amount of nondeterminism. Specifying a fair merge which does not ignore any of its input channels forever, provides a good example. But even then, one may still recourse to a deterministic implementation which ensures that all possibilities will be actualized in the consecutive calls of the function implementing a fair nondeterministic operation.

Finally, one can think of a target programming language which contains nondeterministic constructs (like random number generators, or choice operations). In such a case, the responsibility for "implementing" nondeterminism is delegated to the implementor of the language. The specification formalism only helps developing programs in such a target language.

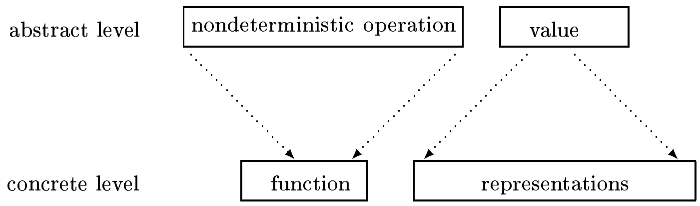
In any case, it seems reasonable to admit restricted degree of nondeterminism in an implementation of a nondeterministic data type as long as such a restriction remains consistent with the original specification.

In this paper, we are interested in extending the classical notions of structural specifications to nondeterministic context. We apply the basic notion of the inclusion of the model classes (with identical signatures) as the implementation relation. It turns out that several notions generalize trivially. Extending the set of axioms, forgetting or renaming some sorts or operations, yield implementations by an obvious analogy to the deterministic case. The central problem is *data refinement* and we study this issue in more detail. We will not consider behavioral equivalence or abstractor implementations (in the sense of [ST88]).

The paper is organized as follows: in section 2 we motivate the need for nondeterministic data types by refering the known problem of implementing sets with choice operation by sequences. Traditionally, to verify this implementation one introduced the notion of behavioral equivalence and implementation. We achieve the same effect by considering the underspecified choice operation as nondeterministic. In section 3 the general setting for writing specifications of nondeterministic data types and its intuitive semantics are introduced. Section 4 illustrates how various formalisms for dealing with nondeterminism may be utilized in this setting, focusing on the aspect of data refinement. Section 5 discusses the verification of the implementation of nondeterministic data types in greater generality. Section 6 introduces the notion of parameterisation as a particular specification-building operation illustrating it with an example.

2 The Choice Problem

The relevance of data refinement for the problem of implementation of nondeterministic data types is the result of the fact that nondeterminism at the abstract level may be a reflection of the existence of different concrete representations of one abstract value. Implementing the abstract value by several concrete representations, one may, simultaneously, restrict the nondeterminism of an abstract operation. In [Sub81] such implementations are called *pseudo-nondeterministic*. Schematically, we may express such a relationship in the following diagram:



This interaction of simultaneous data refinement (of the abstract values) and restricting nondeterminism (of the abstract operations) causes verification problems. One of its classical examples is the attempt to implement sets with a (nondeterministic) choice operation by sequences with the function *fst* returning the head of the list (Figure 1). Axioms $U3, U3'$ follow from the rest of the respective axiom sets only by induction. Therefore we have included them explicitly.

The crucial problem of the choice function $\sqcup$ is as follows: while identity $\doteq$ is a congruence on Set(E), the relation $\equiv$, expressing that two sequences represent the same set, is not a congruence on Seq(E). If we insisted on keeping the axiom $E4$, we would obtain:

$$S\&T \equiv T\&S \;\Rightarrow\; fst(S\&T) \equiv fst(T\&S) \tag{1}$$

which leads to an inconsistency. In fact, it is impossible to define a congruence on Seq(E) which would make SEQ(E) an implementation of SET(E).

The problem has nothing to do with the fact that the choice function $\sqcup$ may be thought of as nondeterministic. In fact, it is exactly the postulated congruence: $S \doteq Q \Rightarrow \sqcup(S) \doteq \sqcup(Q)$ which leads to the undesirable effect: $S \cup T \doteq T \cup S \;\Rightarrow\; \sqcup(S \cup T) \doteq \sqcup(T \cup S)$. Thus, the problem arises rather because $\sqcup$ is considered deterministic and underspecified.

To get rid of this problem, we

- need a syntactic means of distinguishing between deterministic and *possibly* nondeterministic operations, and
- restrict the congruence requirement $E4$ to deterministic operations.

This was suggested already in [QG93] and will apply to all different approaches which we will consider in the following sections. Notice that the second point implies that we introduce a distinction between underspecified and (possibly) nondeterministic operations – the former ones respect congruence.

In the present example, we will consider $\sqcup$ as a nondeterministic operation which is not referentially transparent. In other words, we do not postulate validity of $E4 : S \doteq Q \Rightarrow \sqcup(S) \doteq \sqcup(Q)$. Even the equality $\sqcup(S) \doteq \sqcup(S)$, is not supposed to hold! This releases us from the obligation to prove that $\equiv$ is a congruence with respect to *fst*. But then, although $fst(S) \doteq fst(S)$, it is not the case that $\sqcup(S) \doteq \sqcup(S)$. On the other hand, two occurrences of $\sqcup(S)$ may be seen as equivalent in the sense that they have the same sets of possible results: $\sqcup(S) \asymp \sqcup(S)$. Verification of data refinement will have to consider the fact that nondeterminism of an abstract operation is modeled

SET(E)				σ	**SEQ(E)**			
$\mathcal{S} = \{E, Set(E)\}$					$\mathcal{S}' = \{E, Seq(E)\}$			
$\mathcal{F}$					$\mathcal{F}'$			
$\emptyset :$		$\to$	$Set(E)$	$\mapsto$	$\epsilon :$		$\to$	$Seq(E)$
$\circ :$	$Set(E) \times E$	$\to$	$Set(E)$	$\mapsto$	$\hat{}\ :$	$Seq(E) \times E$	$\to$	$Seq(E)$
$\cup :$	$Set(E) \times Set(E)$	$\to$	$Set(E)$	$\mapsto$	$\& :$	$Seq(E) \times Seq(E)$	$\to$	$Seq(E)$
$\in :$	$E \times Set(E)$	$\to$	$Bool$	$\mapsto$	$\in :$	$E \times Seq(E)$	$\to$	$Bool$
$\setminus :$	$Set(E) \times E$	$\to$	$Set(E)$	$\mapsto$	$\setminus :$	$Seq(E) \times E$	$\to$	$Seq(E)$
$\sqcup :$	$Set(E)$	$\to$	E	$\mapsto$	$fst :$	$Seq(E)$	$\to$	E
$\doteq :$	$Set(E) \times Set(E)$	$\to$	$Bool$	$\mapsto$	$\equiv :$	$Seq(E) \times Seq(E)$	$\to$	$Bool$
					$\doteq :$	$Seq(E) \times Seq(E)$	$\to$	$Bool$

$\mathcal{A}$					$\mathcal{A}'$		
$S \circ x \circ y$	$\doteq$	$S \circ y \circ x$	$A1.$	$A1'.$	$S\hat{}x\hat{}y$	$\equiv$	$S\hat{}y\hat{}x$
$S \circ x \circ x$	$\doteq$	$S \circ x$	$A2.$	$A2'.$	$S\hat{}x\hat{}x$	$\equiv$	$S\hat{}x$
$S \cup \emptyset$	$\doteq$	S	$U1.$	$U1'.$	$S\&\epsilon$	$\doteq$	S
$S \cup (T \circ x)$	$\doteq$	$(S \cup T) \circ x$	$U2.$	$U2'.$	$S\&(T\hat{}x)$	$\doteq$	$(S\&T)\hat{}x$
$S \cup T$	$\doteq$	$T \cup S$	$U3.$	$U3'.$	$S\&T$	$\equiv$	$T\&S$
$x \in \emptyset$	$\doteq$	$\mathbf{ff}$	$M1.$	$M1'.$	$x \in \epsilon$	$\doteq$	$\mathbf{ff}$
$x \in (S \circ y)$	$\doteq$	$(x \doteq y \ \vee x \in S)$	$M2.$	$M2'.$	$x \in (S\hat{}y)$	$\doteq$	$(x \doteq y \ \vee\ x \in S)$
$\emptyset \setminus x$	$\doteq$	$\emptyset$	$D1.$	$D1'.$	$\epsilon \setminus x$	$\doteq$	ϵ
$S \circ x \setminus x$	$\doteq$	S	$D2.$	$D2'.$	$S\hat{}x \setminus x$	$\doteq$	$S \setminus x$
$x \not\doteq y$	$\Rightarrow$	$(S \circ y) \setminus x \doteq (S \setminus x) \circ y$	$D3.$	$D3'.$	$x \not\doteq y$	$\Rightarrow$	$(S\hat{}y) \setminus x \doteq (S \setminus x)\hat{}y$
$\sqcup(S) \in S$	$\vee$	$S \doteq \emptyset$	$C.$	$C1'.$	$fst(\epsilon\hat{}x)$	$\doteq$	x
				$C2'.$	$fst(S\hat{}x\hat{}y)$	$\doteq$	$fst(S\hat{}x)$
				$E0.$	$Q \doteq S$	$\Rightarrow$	$Q \equiv S$
S	$\doteq$	S	$E1.$		Q	$\equiv$	Q
$Q \doteq S$	$\Rightarrow$	$S \doteq Q$	$E2.$		$Q \equiv S$	$\Rightarrow$	$S \equiv Q$
$Q \doteq S, S \doteq P$	$\Rightarrow$	$Q \doteq P$	$E3.$		$Q \equiv S, S \equiv P$	$\Rightarrow$	$Q \equiv P$
$Q \doteq S$	$\Rightarrow$	$f(...Q...) \doteq f(...S...)$	$E4.$		$Q \equiv S$	$\Rightarrow$	$f(...Q...) \equiv f(...S...)$

Figure 1: Specifications of sets and sequences with the intended implementation relation

by the multiple concrete representations of data. For instance, the formula $\sqcup(S \cup T) \asymp \sqcup(T \cup S)$ will be verified not by the equality $fst(S\&T) \doteq fst(T\&S)$ (which does no hold), but by the set-equality $\bigcup_{X \equiv S\&T} fst(X) \asymp \bigcup_{X \equiv T\&S} fst(X)$.

3 The General Set-up

In this section we introduce the syntax and give an intuitive semantics for specifications with nondeterministic operations. We also indicate the general procedure for interpreting such specifications using some available formalism dealing with nondeterminism. The precise semantics of the specification will depend on the choice of such a formalism, and will be discussed in the next section. Finally, we indicate the procedure for the verification of data refinement.

First, we extend the signature Σ to a triple $\langle \mathcal{S}, \mathcal{F}, \mathcal{N} \rangle$, where $\mathcal{N}$ is a set of operations symbols which may be (and are treated as) nondeterministic. We always assume that $\mathcal{F}$ is non-empty. This reflects the intuition that nondeterminism may occur in a specification but, typically, will be only a minor part of it – most of the operations will still be deterministic and defined in the usual fashion. The congruence axiom $E4$ applies only to functions in $\mathcal{F}$ but not to the operations from $\mathcal{N}$. These operations are defined in the usual (equational) manner. To make them explicit, we will use the letter N with a subscript indicating the actual operation. For instance, the specification of sets with a choice operation is obtained by taking the set axioms but changing the signature and the choice axiom C as follows:

$$
\begin{array}{lll}
\text{SET(E) } \mathbf{is} & (...as\ before\ except\ for\ C...) \\
& \mathcal{N}: \quad \sqcup: \quad Set(E) \quad \rightarrow \quad E \\
& \mathcal{A}: \quad \mathbf{N}_{\sqcup}(S) \in S \qquad \vee \quad S \doteq \emptyset
\end{array}
$$

The axiom is intended to mean that *any* result of a choice from a non-empty set S belongs to S. We will present the general strategy, common to all the approaches considered in the later section¡s, using the informal relation $r \leftharpoondown f$ expressing that "r is a possible result of N_f". Thus, the above axiom says: $\forall r \leftharpoondown \sqcup(S) : r \in S \vee S \doteq \emptyset$.[2]

There is an important distinction between the *set of all* results and a *particular* one. Typically, one defines a nondeterministic operation in terms of another one, that is, *defines any* possible result of one operation by *using one particular* result of another one. For instance, an operation $rem(S)$ removing an arbitrary element from a set S could be defined using the choice operation as follows: *any* result of $rem(S)$ is obtained by choosing *some* element x from S and removing it from S. This distinction between the occurrences of nondeterministic operations being *defined* and those being *used* will be reflected in using the **bold** font for the former and the usual font for the latter. The defined occurrences correspond to universal and the used occurrences to existential quantification over the possible results returned by the operation. Writing $\mathbf{N}_f$ we are referring to the whole set of possible results (or more precisely, to an arbitrary element of this set), while N_f refers to a particular element of this set.

The general rules for translating the specifications are then as follows:

$$
\begin{array}{rcl}
\mathbf{N}_f(x) \doteq t & \stackrel{\nu}{\Longrightarrow} & \forall r \leftharpoondown f(x) : r \doteq t \\
N_f(x) \doteq t & \stackrel{\nu}{\Longrightarrow} & \exists r \leftharpoondown f(x) : r \doteq t \\
\mathbf{N}_f(x) \doteq N_g(x) & \stackrel{\nu}{\Longrightarrow} & \forall r_1 \leftharpoondown f(x)\ \exists r_2 \leftharpoondown g(x) : r_1 \doteq r_2
\end{array}
\tag{2}
$$

These rules (as well as all other translation schema given in the following) are defined only for the atomic equations and may be applied to any equation-based specification language: for equational specifications, for conditional equations, for sequents, clauses, etc. In all our examples, the variables in the original specifications are (implicitly) universally quantified. We will also assume that universal and existential occurrences are not nested within each other.

Remark. It seems straightforward to formulate these and following rules allowing such a nesting. For instance, one would translate $N_f(\mathbf{N}_g(N_h(x)))$... into $\exists r_1 \leftharpoondown h(x)\ \forall r_2 \leftharpoondown g(r_1)\ \exists r_3 \leftharpoondown f(r_2)$...

However, this would make all the formulations more intricate and, more importantly, we have not encountered any examples where such a nesting would be necessary. Therefore we make this simplifying assumption.

If t is a deterministic term, the first formula in (2) expresses determinacy of $N_f(x)$. It is only the second formula which may be used to demand that a nondeterministic operation is truly nondeterministic. For instance, writing two axioms $N_g(a) = 1$ and $N_g(a) = 2$, we make an explicit demand that both 1 and 2 should be among the possible results of $g(a)$. The third formula allows us to make the defined operation *at most* as nondeterministic as the used one. $\mathbf{N}_f(a) = N_g(a)$ does not force $f(a)$ to return both 1 and 2, but only one of the two. This setting reflects our interest in describing the "upper bound" of nondeterministic operations without introducing any implicit assumptions (typically, of a semantic nature) which would imply a kind of "maximal nondeterminacy by default", as it happens in the initial approach to the semantics of nondeterminism [Hus93], [Mos89], [Mes92].

Remark. Alternatively, we could choose to interpret all occurrences of N_f existentially. But this would be too permissive – axiom 1., for instance, would then mean: $\exists s : s \leftharpoondown \sqcup(S) \wedge s \in S$. We want to ensure that $s \in S$ holds for all s results of $\sqcup(S)$.

As a more elaborate example, illustrating also the full use of this language, consider a definition of the depth-first traversal of a directed graph, which returns a DFS-tree. Let a graph be given (we assume it to be fixed for an instance of DFS to avoid repeating inessential arguments) as a set of edges (each edge E being a pair of vertices $V \times V$), with a function $ch : V \rightarrow SET(V)$

[2] We use this abbreviation for bounded quantification $\forall r : r \leftharpoondown \sqcup(S) \Rightarrow r \in S$... Similarly, we will write $\forall r \equiv s : \Phi(r)$. For existential quantifiers, the abbreviation $\exists r \equiv s : \Phi(r)$ stands for $\exists r : r \equiv s \wedge \Phi(r)$.

which, for each vertex v, returns the set of its children (adjacent vertices). The (nondeterministic) operation *dfs* starting from a vertex v calls the auxiliary operation *tr* (ax.2) which traverses the graph from v by visiting recursively all vertices in $ch(v)$. Its last argument T is the tree built so far, which is returned as the result in the moment when the second argument (the set of children remaining to be visited) becomes empty (ax.3). The vertex to be visited next among the children in the second argument, is chosen nondeterministically. If it has been visited before (the Boolean test $x \sqsubset T$ returns $\mathbf{tt}$, if x is a vertex in the graph T), it is removed from the set (ax.4). Otherwise, *tr* descends recursively and continues traversing the graph from this vertex, removing it from the current set and adding the corresponding edge to the result tree (ax.5).

$$
\begin{array}{llrcl}
\text{DFS } \textbf{is} & \multicolumn{4}{l}{\textbf{enrich } SET(V), SET(E) \textbf{ by}} \\
 & \mathcal{N}: & \mathit{dfs}: \quad V & \rightarrow & Set(E) \\
 & & \mathit{tr}: \quad V \times Set(V) \times Set(E) & \rightarrow & Set(E) \\
 & \mathcal{A}: & \mathbf{N}_{\mathit{dfs}}(v) & \doteq & N_{tr}(v, ch(v), \emptyset) \\
 & & \mathbf{N}_{tr}(v, \emptyset, T) & \doteq & T \\
 & & S \not\doteq \emptyset \wedge x \doteq N_{\sqcup}(S) \wedge x \sqsubset T & \Rightarrow & \mathbf{N}_{tr}(v, S, T) \doteq N_{tr}(v, S \setminus x, T) \\
 & & S \not\doteq \emptyset \wedge x \doteq N_{\sqcup}(S) \wedge x \not\sqsubset T & \Rightarrow & \mathbf{N}_{tr}(v, S, T) \doteq \\
 & & & & \quad N_{tr}(v, S \setminus x, N_{tr}(x, ch(x), T \circ \langle v, x \rangle))
\end{array}
$$

We allow one to use the N_f expressions like usual functions and compose them with other operations. We may write $\emptyset \circ N_{\sqcup}(S)$ – a set with one element chosen from S, or $N_{\sqcup}(S \setminus N_{\sqcup}(S))$ – an element chosen arbitrarily from the set S without an arbitrary element. Although we do not assume, in general, that these operations satisfy the congruence axiom, one may still express such a claim: $x \doteq y \Rightarrow \mathbf{N}_f(x) \doteq \mathbf{N}_f(y)$. As was explained above, such an axiom would force $f(x)$ to be deterministic, since it amounts to saying: $x \doteq y \Rightarrow \forall r \leftharpoonup f(x)\ \forall s \leftharpoonup f(y) : r \doteq s$.

Observe that the above specification follows exactly the natural prescription to be found in any standard algorithm book. This naturalness, however, is heavily dependent on the fact that the operation *dfs* is treated as nondeterministic. It does not return *the* DFS-tree but *an arbitrary* DFS-tree. If we tried to make N_{dfs}, N_{tr} deterministic, we would obtain unsound result – the equation $\mathbf{N}_{\mathit{dfs}}(v) \doteq \mathbf{N}_{\mathit{dfs}}(v)$ would identify all possible DFS-trees. This problem is discussed in more detail in [WM95b].

Our general strategy is now to transform axioms of the specifications written with the above formalism into formulas of standard (first- or higher order) logic and to carry out the verification of implementation at this level. Applying the scheme (2), we obtain the following interpretation of the choice and DFS-axioms:

$$
\begin{array}{lrcl}
1. & \forall S : (\forall r \leftharpoonup \sqcup(S) : r \in S) & \vee & S \doteq \emptyset \\
2. & \forall v,\ \forall r_1 \leftharpoonup \mathit{dfs}(v)\ \exists r_2 \leftharpoonup tr(v, ch(v), \emptyset) & : & r_1 \doteq r_2 \\
3. & \forall v, T,\ \forall r \leftharpoonup tr(v, \emptyset, T) & : & r \doteq T \\
4. & \forall S, T, x, v : S \not\doteq \emptyset \wedge (\exists r \leftharpoonup \sqcup(S) : x \doteq r) \wedge x \sqsubset T & \Rightarrow & \forall r_1 \leftharpoonup tr(v, S, T) \\
 & & & \exists r_2 \leftharpoonup tr(v, S \setminus x, T) : r_1 \doteq r_2 \\
5. & \forall S, T, x, v : S \not\doteq \emptyset \wedge (\exists r \leftharpoonup \sqcup(S) : x \doteq r) \wedge x \not\sqsubset T & \Rightarrow & \forall r_1 \leftharpoonup tr(v, S, T) \\
 & & & \exists r_2 \leftharpoonup tr(x, ch(x), T \circ \langle v, x \rangle) \\
 & & & \exists r_3 \leftharpoonup tr(v, S \setminus x, r_2) : r_1 \doteq r_3
\end{array}
\qquad (3)
$$

This procedure still presupposes that we have syntactic means to describe the relation $\leftharpoonup$ which, in turn, requires means of talking about the sets occurring on the right-hand side of $\leftharpoonup$. All four approaches mentioned in the introduction provide such means which yield results corresponding to (3). Thus, instead of fixing some particular way of doing that, we will examine all of them indicating the associated translation scheme. In this way, we obtain a two level translation: first from our specification to one of the known formalisms, and then to the standard logic, as illustrated in the upper part of Figure 2 below. The translation τ is used when verifying data refinement. Because refinement of nondeterministic data types may resolve some amount of nondeterminism in favor of a (more) deterministic implementation, verification of abstract axioms has to take this fact into account.

$$\text{original specification} \xrightarrow{\nu} \text{a formalism for nondeterminism} \xrightarrow{\delta} \text{standard logic}$$

$$\begin{tikzcd}
SP \arrow[r, "\nu"] \arrow[d, "\leadsto"] & \nu(\mathcal{A}) \arrow[r, "\delta"] & \delta(\nu(\mathcal{A})) \arrow[d, "\Rightarrow"] & \\
SP' \arrow[r, "\nu"] \arrow[d, "\overset{\equiv}{\leadsto}"] & \nu(\mathcal{A}') \arrow[r, "\delta"] & \delta(\nu(\mathcal{A}')) \arrow[dr, "\Rightarrow"] & \\
SP'' \arrow[r, "\nu"] & \nu(\mathcal{A}'') \arrow[r, "\delta"] & \delta(\nu(\mathcal{A}'')) \arrow[r, "\tau"] & \tau(\delta(\nu(\mathcal{A}'')))
\end{tikzcd}$$

Figure 2: Translation of nondeterminism into standard logic

In all cases the translation procedure is quite analogous (although the results are not), and follows the lines of (2). We may think of it as being parameterised by a particular formalism for treating nondeterminism. We only have to assume that the translation ν is compositional with respect to the logical operators which we are using in our specifications, and that so is δ.

The advantage of such an approach is that in the cases when the intermediary formalism provides some means of verifying implementations, we will be free to use it. Of course, since various approaches to nondeterminism we are considering are not semantically equivalent, we assume the choice being fixed for the whole development process.

3.1 Nondeterministic Data Refinement

Before giving the translation rules for the different formalisms, we indicate the main idea of verification of data refinement, using the intuitive relation $\leftharpoonup$. We formulate the proof obligations in terms of some standard logical formalism on the basis of a translation τ of the abstract axioms. Although such a translation may lead to rather complicated formulas, we still consider it a fruitful approach. Its main focus is on the easy to use the top-level formalism, which provides the basis for nondeterminism without interfering with the standard parts of the specification. In a practical development process, verification is carried out only in some, most critical cases. Formulating the verification conditions in a standard (for the most first-order) logic, makes it possible to apply all existing tools for (semi-)automatic verification of these critical steps.

For the rest of this and the next section we focus on the data refinement. A more general implementation relation is studied in section 5. We restrict our attention to the $SET(E)$ with choice operation (axiom 1. above) which is to be implemented by $SEQ(E)$ with function *fst*.

Because nondeterministic operations in $\mathcal{N}$ do not respect the congruence with respect to element-equality $\doteq$, we will not require their implementations to satisfy the congruence axiom $E4$ with respect to the implementation $\equiv$ of $\doteq$. $\equiv$ will be a congruence only with respect to the implementation of deterministic functions. However, we have to consider the fact that nondeterministic operations respect congruence with respect to set-equality. For instance, we have:

$$S \doteq T \wedge x \doteq N_{\sqcup}(S) \;\Rightarrow\; x \doteq N_{\sqcup}(T)$$

This is not a congruence with respect to element-equality $\doteq$ but with respect to set-equality $\asymp$. Translating it with ν from (2), we get

$$S \doteq T \wedge (\exists r_1 \leftharpoonup \sqcup(S) : x \doteq r_1) \;\Rightarrow\; \exists r_2 \leftharpoonup \sqcup(T) : x \doteq r_2 \tag{4}$$

which means: $S \doteq T \Rightarrow \sqcup(S) \asymp \sqcup(T)$.

The problematic point in this context is that data refinement may remove some nondeterminism. Since one set S may have multiple representations as sequences, the nondeterminism of $\sqcup$ may best be understood as a "lack of knowledge" as to which, among the equivalent representations of S, is being used at the moment. At the abstract level a possible result of an operation is modeled by the nondeterminism of the operation. At the level of implementation a possible result may reflect either (some left) nondeterminism or a particular choice of the representation for the abstract argument. Verification of the above formula should not take the form $S \equiv T \wedge x \doteq fst(S) \Rightarrow x \doteq fst(T)$. Instead, we should consider the possible results returned by fst when applied to the whole equivalence class of S:

$$S \equiv T \wedge (\exists S' \equiv S : x \doteq fst(S')) \;\Rightarrow\; \exists T' \equiv T : x \doteq fst(T') \tag{5}$$

Notice the similarity between the formulas (5) and (4).

The verification of the implementation of $SET(E)$ by $SEQ(E)$ boils then down to the standard verification of the implementation of the deterministic functions, and checking that axioms of $SEQ(E)$ imply the formula:

$$\forall S, S', x : S \equiv S' \wedge x \doteq fst(S) \Rightarrow x \in S'$$

To verify a data refinement, we have to show that the implementation implies the abstract axioms transformed with a translation function τ which replaces each term of the refined sort by its equivalence class, and where the quantification over elements of such classes reflects the quantification from (2). Each occurrence of $N_f : \ldots S \ldots \to T$, where S is the refined sort is replaced according to the following scheme:

$$\begin{array}{rcl} \mathbf{N}_f(...x...) \doteq t & \stackrel{\tau}{\Longrightarrow} & \forall y \equiv x \; \forall r \leftharpoondown f(...y...) : r \equiv t \\ N_f(...x...) \doteq t & \stackrel{\tau}{\Longrightarrow} & \exists y \equiv x \; \exists r \leftharpoondown f(...y...) : r \equiv t \\ \mathbf{N}_f(...x...) \doteq N_g(...x...) & \stackrel{\tau}{\Longrightarrow} & \forall y \equiv x \; \forall r_1 \leftharpoondown f(...y...) \; \exists z \equiv x \; \exists r_2 \leftharpoondown g(...z...) : r_1 \equiv r_2 \end{array} \tag{6}$$

Occurrences of (sub)terms of the refined sort in N_f will be called *existential* (for instance, x in the second line, the second occurrence of x in the third line); occurrences in $\mathbf{N}_f$ are called *universal.* Because of our convention, we do not have mutual nesting of universal and existential occurrences, and hence it is the whole side of an equation which is either universal or existential.

Also, notice that x, which occurs free in the LHS (left-hand side), remains free in the RHS. If there are n occurrences of a term t of refined sort in the scope of a nondeterministic operation in the LHS, we will introduce n new variables $t_1 ... t_n$ which all are appropriately quantified and equivalent to t in the RHS.

The above rules illustrate only the replacement of immediate subterms but, of course, τ requires replacement of all refined subterms of nondeterministic terms. If we have an operation $N_f : S \times E \to S$ and an axiom involving $N_f(N_f(s,x),y)...$, we will get $\exists s_0 \equiv s \; \exists r_1 \leftharpoondown f(s_0,x) \; \exists s_1 \equiv r_1 \; \exists r_2 \leftharpoondown f(s_1,y) : r_2 ...$. In general, let $[x] = \{y : y \equiv x\}$. Since $\equiv$ is not a congruence (for nondeterministic operations), $[f([t])] \neq [f(t)]$, we have to keep the parantheses $[\_]$ at all subterms of the refined sort. Translation τ will introduce a new (quantified) variable for each subterm embraced by $[\_]$ occurring within a nondeterministic term. This is summarized in the table below (Figure 3). Then, each $[t]$ within the scope of a nondeterministic operation in an atomic ϕ is replaced by

τ	is of a refined sort	is not of a refined sort
$x \in \mathcal{V}$	$[x]$	x
$c : \to S$	$[c]$	c
$f(t_1 \ldots t_n)$	$[f(\tau(t_1) \ldots \tau(t_n))]$	$f(\tau(t_1) \ldots \tau(t_n))$

Figure 3: Replacement of nested terms of refined sort

$Qx \equiv t : \phi^t_x$ (where Q is an appropriate quantifier), respectively by $Qx \leftharpoondown t, Qx_1 \equiv x : \phi^t_{x_1}$ if t is nondeterministic.

4 Using Various Formalisms for Nondeterminism

In this section we show how we can translate our specifications into various formalisms designed for an explicit treatment of nondeterminism. To give the full flavor, we show the translation of all choice and DFS-axioms. To illustrate the resulting proof obligations for data refinement, we use only the example of sets with the choice operation.

The following subsections (treating, respectively, relations, multialgebras, sets of functions and oracles) have all the same structure including: 1) the definition of the interpretation function ν and the DFS example, 2) the proof obligations for data refinement expressed in terms of the translation τ, and 3) the semantic construction (quotient) together with the theorem stating the correctness of the verification criteria with respect to this construction. The last part is missing in the case of oracles for the reasons explained in the last subsection.

4.1 Relations

The relational approach to specification is not very widely used. The works like [BGW80], [BW81], [Sta89] point in this direction and [Hoo92], [Voe91] propose a formalism based on the relations as the fundamental concept. The realtional specifications tend to be very unreadable, albeit concise, and we would not try to apply a full, abstract relational language but make only very simple use of relations in this section.

$x \leftharpoonup f(y)$ defines a relation between x and y, and is represented directly in the relational language as $R_f(y,x)$. In general, an operation $N_f : S_1 \times ... \times S_n \to S$ is modeled as a relation $R_f \subseteq S_1 \times ... \times S_n \times S$, that is, one implicitly assumes that the last argument of the relation corresponds to the result returned by the operation. Hence, we need a special definition of relational composition which would reflect the composition of operations. Given two relations $R_1 \subseteq S_1 \times ... \times S_n \times S$ and $R_2 \subseteq S \times T_1 \times ... \times T_m \times T$, we denote by $R_2(R_1)$ their relational composition on the S argument. More precisely, $R_2(R_1) = \{\langle s_1...s_n, t_1...t_m, t\rangle : \exists s : R_1(s_1...s_n, s) \land R_2(s, t_1...t_m, t)\}$.

Then, the translation ν into relational specifications looks as follows:

$$\begin{array}{rcl} \mathbf{N}_f(s) \doteq t & \stackrel{\nu}{\Longrightarrow} & \forall x : R_f(s,x) \Rightarrow x \doteq t \\ N_f(s) \doteq t & \stackrel{\nu}{\Longrightarrow} & \exists x : R_f(s,x) \land x \doteq t \\ \mathbf{N}_f(s) \doteq N_g(s) & \stackrel{\nu}{\Longrightarrow} & \forall x\ \exists y : R_f(s,x) \Rightarrow (R_g(s,y) \land x \doteq y) \end{array} \tag{7}$$

Remark. Instead, in the second and third line we could dispense with the quantifiers and get $R_f(s,t)$, respectively, $\forall x : R_f(s,x) \Rightarrow R_g(s,x)$. Then, the only case when existential quantifiers have to be retained would be existential equality: $N_f(x) \doteq N_g(x)$, which corresponds to $\exists r_1 \leftharpoonup f(x) \exists r_2 \leftharpoonup g(x) : r_1 \doteq r_2$, is translated as $\exists r_1, r_2 : R_f(x,r_1) \land R_g(x,r_2) \land r_1 \doteq r_2$. We do not focus on this kind of equality because its use and importance seem to be very limited. On the other hand, because the axioms may involve nested applications of N_f, the translation may involve relational composition as defined above, which introduces additional existential quantifiers.

We will keep the quantifiers as in (7) because this will make the definition of the translation τ for data refinement depend only on the result of ν. Otherwise, we would have to refer back to the original specification in order to identify existential and universal occurrences.

The translated axioms for choice and DFS look as follows:

$$\begin{array}{lrcl} 1. & (\forall x : R_{\sqcup}(S,x) \Rightarrow x \in S) & \vee & S \doteq \emptyset \\ 2. & R_{dfs}(v,r) & \Rightarrow & R_{tr}(v, ch(v), \emptyset, r) \\ 3. & R_{tr}(v,\emptyset,T,r) & \Rightarrow & r \doteq T \\ 4. & S \not\doteq \emptyset \land R_{\sqcup}(S,x) \land x \sqsubset T & \Rightarrow & R_{tr}(v,S,T,r) \Rightarrow R_{tr}(v, S \setminus x, T, r) \\ 5. & S \not\doteq \emptyset \land R_{\sqcup}(S,x) \land x \not\sqsubset T & \Rightarrow & R_{tr}(v,S,T,r) \Rightarrow \exists M : R_{tr}(x, ch(x), T \circ \langle v,x\rangle, M) \\ & & & \qquad \land\ R_{tr}(v, S \setminus x, M, r) \end{array}$$

Equivalently, we can gather all the defined occurrences of a relation into one formula:

$$
\begin{array}{llll}
3. & R_{tr}(v,S,T,r) & \Rightarrow & S \doteq \emptyset \Rightarrow r \doteq T \ \wedge \\
4. & & & S \not\doteq \emptyset \wedge R_{\sqcup}(S,x) \wedge x \sqsubset T \Rightarrow R_{tr}(v, S \setminus x, T, r) \ \wedge \\
5. & & & S \not\doteq \emptyset \wedge R_{\sqcup}(S,x) \wedge x \not\sqsubset T \ \Rightarrow \exists M : R_{tr}(x, ch(x), T \circ \langle v, x \rangle, M) \\
& & & \qquad\qquad\qquad\qquad\qquad\qquad \wedge\ R_{tr}(v, S \setminus x, M, r)
\end{array}
$$

However, we do not define the relation fully, since the above translation introduces implications from the defined occurrences, and we do not replace them by the equivalence relation. This reflects the intended interpretation described in (2). Of course, among the models of the above specification there will be the ones where $R_{\sqcup}(S,x) \Leftrightarrow x \in S$.

4.1.1 The Proof Obligations for Data Refinement

Relational specifications introduce directly the language of first-order logic in which we intend to formulate the refinement conditions. Thus we have $\delta = id$.

The translation τ of the abstract axioms is defined according to (6). We indicate the translation as if all (sub)terms were of the refined sort. The first line gives the general schema and the second (tiny) one the special case when R_f is a function f:

$$
\begin{array}{rcl}
\forall x : R_f(s,x) \Rightarrow x \doteq t & \stackrel{\tau}{\Longrightarrow} & \forall s_1 \equiv s\ \forall x, x_1 \equiv x : R_f(s_1, x_1) \Rightarrow x \equiv t \\
& & {\scriptstyle \forall s_1 \equiv s\ :\ f(s_1) \equiv t} \\
\exists x : R_f(s,x) \wedge x \doteq t & \stackrel{\tau}{\Longrightarrow} & \exists s_1 \equiv s\ \exists x, x_1 \equiv x : R_f(s_1, x_1) \wedge x \equiv t \\
& & {\scriptstyle \exists s_1 \equiv s\ :\ f(s_1) \equiv t} \\
\forall x\ \exists y : R_f(s,x) \Rightarrow (R_g(s,y) \wedge x \doteq y) & \stackrel{\tau}{\Longrightarrow} & \forall x, x_1 \equiv x, s_1 \equiv s\ \exists y, y_1 \equiv y\ \exists s_2 \equiv s : \\
& & \qquad R_f(s_1, x_1) \Rightarrow (R_g(s_2, y_1) \wedge x \equiv y) \\
& & {\scriptstyle \forall s_1 \equiv s\ \exists s_2 \equiv s\ :\ f(s_1) \equiv g(s_2)}
\end{array}
\tag{8}
$$

Instead of t in the first two formulas, we should have used $\forall t_1 \equiv t ... t_1$. But this is not necessary since $\equiv$ is an equivalence relation. Notice, however, that since we assumed the result sort (variables x and y) to be refined, we cannot dispense with the introduction and quantification over the respective new variables x_1 and y_1. For instance, the first formula is not equivalent to $\forall s_1 \equiv s\ \forall x : R_f(s_1, x) \Rightarrow x \equiv t$. As emphasized after the schema (6), *each* (sub)term of the refined sort within the scope of a nondeterminisitic operation, will give rise to a new variable. This rule applies also to the whole nondeterministic term which after translation ν corresponds to the result variable (x in this formula).

4.1.2 Semantics

The semantics of relations may be given in the standard way by (first-order) structures with predicates. The semantics of quotient construction with respect to an equivalence $\equiv$ is a direct generalization of the standard case where $\equiv$ is a congruence.

Definition 4.1 Given a (relational) Σ-strucutre A and an equivalence relation $\equiv$, let $[x]$ for an $x \in |A|$ denote the equivalence class $\{y : y \equiv x\}$. The *quotient structure* $A/_{\equiv}$ is defined as follows:

- $|A/_{\equiv}| = \{[x] : x \in |A|\}$
- $c^{A/_{\equiv}} = [c]$ for each $c \in \mathcal{F}$
- $f^{A/_{\equiv}}([x_i]) = [f^A(x_i)]$ for each $f \in \mathcal{F}$
- $R^{A/_{\equiv}}([x_i]) \Leftrightarrow \exists x_i' \in [x_i] : R^A(x_i')$ for each $R \in \mathcal{N}$

Because $\equiv$ is a congruence with respect to deterministic operations, the third point gives a well-defined $f^{A/_{\equiv}}$. The last point corresponds to the homomorphism condition: even if x, y are such that $\neg R^A(x,y)$, then the existence of $x' \equiv x$ and $y' \equiv y$ such that $R^A(x', y')$ suffices to make $R^{A/_{\equiv}}([x],[y])$.[3] This latter relation embodies the full nondeterminism of different representations.

[3] It is easy to check that the mapping defined by $x \mapsto [x]$ is a homomorphism from A to $A/_{\equiv}$.

Since a function is just a special case of relation the definition makes sense also in the situations when a relation is implemented by a function.

Given an (implementing) specification with axioms $\mathcal{A}'$ and the (implemented) specification with axioms $\mathcal{A}$, we want to verify that the quotients of models of $\mathcal{A}'$ are models of $\mathcal{A}$: if $A \models \mathcal{A}'$ then $A/_{\equiv} \models \mathcal{A}$. This is verified following the schema from Figure 2 (page 8) by checking that the models of $\mathcal{A}'$ satisfy $\tau(\mathcal{A})$, i.e. that $\mathcal{A}' \Rightarrow \tau(\mathcal{A})$. The correctness of these verification criteria defined by τ is expressed in the following theorem.

Theorem 4.2 $A \models \tau(\mathcal{A}) \iff A/_{\equiv} \models \mathcal{A}$.

Proof. If we forget the relation part, the quotient construction and verification for the functions are the same as in the deterministic case. We show the theorem for the atomic formulas ϕ affected by τ.

1. Let $\phi = \forall x : R_f(s,x) \Rightarrow x \doteq t$:

   $A \models \tau(\phi) \iff A \models \forall x, x_1 \equiv x, s_1 \equiv s : R_f(s_1, x_1) \Rightarrow x \equiv t \iff$
   $A \models \forall x : (\exists x_1 \equiv x, s_1 \equiv s : R_f(s_1,x_1)) \Rightarrow x \equiv t \iff A/_{\equiv} \models \forall [x] : R_f([s],[x]) \Rightarrow [x] \doteq [t] \iff$
   $A/_{\equiv} \models \phi$
2. Let $\phi = \exists x : R_f(s,x) \wedge x \doteq t$:

   $A \models \tau(\phi) \iff A \models \exists x, x_1 \equiv x, s_1 \equiv s : R_f(s,x) \wedge x \equiv t \iff$
   $A/_{\equiv} \models \exists [x] : R_f([s],[x]) \wedge [x] \doteq [t] \iff A/_{\equiv} \models \phi$
3. Let $\phi = \forall x\ \exists y : R_f(s,x) \Rightarrow (R_g(s,y) \wedge x \doteq y)$:

   $A \models \tau(\phi) \iff$
   $A \models \forall x, x_1 \equiv x, s_1 \equiv s\ \exists y, y_1 \equiv y\ \exists s_2 \equiv s : R_f(s_1,x_1) \Rightarrow (R_g(s_2,y_1) \wedge x \equiv y) \iff$
   $A \models \forall x\ \exists y : (\exists x_1 \equiv x, s_1 \equiv s : R_f(s_1,x_1)) \Rightarrow \exists y_1 \equiv y, s_2 \equiv s : R_g(s_2,y_1) \wedge x \equiv y \iff$
   $A/_{\equiv} \models \forall x\ \exists y : R_f([s],[x]) \Rightarrow (R_g([s],[y]) \wedge [x] \doteq [y]) \iff A/_{\equiv} \models \phi$

□

Relations may be the most suggestive concept for modeling nondeterministic operations.

4.2 Multialgebras

Multialgebras have been used by several authors, for instance, in [Hes88], [Hus93], [WM95b]. Here we are following this last work and [Wal93], [WM95c]. Multialgebras work with set-valued functions. Set-valued functions are isomorphic to relations, and so much of the arguments in the current section are similar to those in the previous one. Existential quantifiers correspond in the language of multialgebras to the right-hand side of the inclusion relation $\prec$. $a \prec b$ reads as "a is a possible result of b" (or in general, the results of a are among the results of b). This means that the existential equality is not expressible in this language. In the full version of the language [KW94] one has the operation expressing non-empty intersection which is the exact counterpart of existential equality.

The translation into the multialgebraic language is given by the following rules:

$$\begin{array}{rcl} \mathbf{N}_f(x) \doteq t & \stackrel{\nu}{\Longrightarrow} & f(x) \doteq t \\ t \doteq N_f(x) & \stackrel{\nu}{\Longrightarrow} & t \prec f(x) \\ \mathbf{N}_f(x) \doteq N_g(x) & \stackrel{\nu}{\Longrightarrow} & f(x) \prec g(x) \end{array} \tag{9}$$

The specification of choice and DFS reads then as follows:

$$\begin{array}{lrcl} 1. & \sqcup(S) \in S & \vee & S \doteq \emptyset \\ 2. & dfs(v) & \prec & tr(v, ch(v), \emptyset) \\ 3. & tr(v, \emptyset, T) & \doteq & T \\ 4. & S \not\doteq \emptyset \wedge x \prec \sqcup(S) \wedge x \sqsubset T & \Rightarrow & tr(v,S,T) \prec tr(v, S \setminus x, T) \\ 5. & S \not\doteq \emptyset \wedge x \prec \sqcup(S) \wedge x \not\sqsubset T & \Rightarrow & tr(v,S,T) \prec tr(v, S \setminus x, tr(x, ch(x), T \circ \langle v, x \rangle)) \end{array}$$

The nondeterministic terms in a multialgebraic language are interpreted as (non-empty) sets of possible results, and hence may be treated as predicates over individual values of a given interpretation (carrier of an algebra) A. A term t is understood as a predicate $t^A(\underline{r}) \Leftrightarrow \underline{r} \leftharpoondown t^A \Leftrightarrow \underline{r} \in t^A$. A non-ground term $f(x)$ is a "parameterised" predicate: for each value x, $f^A(x)(\underline{r}) \Leftrightarrow \underline{r} \in f^A(x)$. Thus, terms are translated into formulas (predicates), equality $\doteq$ corresponds to the equivalence of the predicates which, in addition, are satisfied by a unique element, and inclusion $\prec$ corresponds to the implication. The translation δ is:

$$\begin{array}{rcl} f(x) \doteq t & \stackrel{\delta}{\Longrightarrow} & \forall \underline{z} : f(x)(\underline{z}) \Rightarrow \underline{z} \doteq t \\ t \prec f(x) & \stackrel{\delta}{\Longrightarrow} & f(x)(t) \\ f(x) \prec g(x) & \stackrel{\delta}{\Longrightarrow} & \forall \underline{z} : f(x)(\underline{z}) \Rightarrow g(x)(\underline{z}) \end{array} \tag{10}$$

As in the case of relations, the last two lines may be given more elaborate formulations: $\exists \underline{z} : f(x)(\underline{z}) \wedge \underline{z} \doteq t$ and $\forall \underline{z}\ \exists \underline{v} : f(x)(\underline{z}) \Rightarrow (g(x)(\underline{v}) \wedge \underline{z} \doteq \underline{v})$. The reason for not making such a simplification for relations does not apply here, because the translation τ is defined not from the result of δ (10) but of ν (9) which contains the necessary information about existential vs. universal occurrences.

If some $f(x)$ happens to be a function, we may dispense with the introduction of the result variable $\underline{z}$, and obtain simple equalities in all cases.

Remember that terms with nondeterministic subterms are also nondeterministic. If f and g are, respectively, nondeterministic and deterministic, then $g(f(x))$ will also be considered as a predicate: $g(f(x))(\underline{z}) \Leftrightarrow {}^{\forall}/_{\exists}\ \underline{z}_1 : f(x)(\underline{z}_1)\ {}^{\Rightarrow}/_{\wedge}\ g(\underline{z}_1) = \underline{z}$, where quantification depends on whether the occurrence of the whole term is universal or existential.

4.2.1 The Proof Obligations

Let $[x] = \{y : y \equiv x\}$ and extend this notation to sets: $[t] = \{[x] : x \in t\}$. We adapt this notation to the predicates (terms) writing $[t]$ for the set of equivalence classes of the elements satisfying the predicate t. That is, if f is nondeterministic (a predicate $f(\underline{z})$), we define the lifted predicate for:

$$\begin{array}{llcrcl} 1. & f(\underline{z}) & : & [f]([\underline{z}]) & \Longleftrightarrow & \exists \underline{z}_1 \equiv \underline{z} : f(\underline{z}_1) \\ 2. & f(x)(\underline{z}) & : & f([x])(\underline{z}) & \Longleftrightarrow & \exists x_1 \equiv x : f(x_1)(\underline{z}) \\ 3. & f(x)(\underline{z}) & : & [f([x])]([\underline{z}]) & \stackrel{1}{\Longleftrightarrow} & \exists \underline{z}_1 \equiv \underline{z} : f([x])(\underline{z}_1) \\ & & & & \stackrel{2}{\Longleftrightarrow} & \exists \underline{z}_1 \equiv \underline{z}, x_1 \equiv x : f(x_1)(\underline{z}_1) \end{array} \tag{11}$$

The translation τ is obtained by combining the translation δ with (11). If a predicate $f(x)(\underline{z})$ happens to be a function, we may remove $\underline{z}$ and quantification over it, and substitute $f(x)$ for all occurrences of $\underline{z}$ in the translated formula. This is indicated in the last (tiny) line of each case.

$$\begin{array}{rcl} f(x) \doteq t & \stackrel{\tau}{\Longrightarrow} & [f([x])] \doteq [t] \stackrel{\delta}{\Longrightarrow} \forall [\underline{z}] : [f([x])]([\underline{z}]) \Rightarrow [z] \doteq [t] \\ & \stackrel{(11)}{\Longrightarrow} & \forall \underline{z} : (\exists \underline{z}_1 \equiv \underline{z}, x_1 \equiv x : f(x_1)(\underline{z}_1)) \Rightarrow \underline{z} \equiv t \\ & \Longleftrightarrow & \forall \underline{z}, \underline{z}_1 \equiv z, x_1 \equiv x : f(x_1)(\underline{z}_1) \Rightarrow \underline{z} \equiv t \\ & & {\scriptstyle \forall x_1 \equiv x\ :\ f(x_1) \equiv t} \\ t \prec f(x) & \stackrel{\tau}{\Longrightarrow} & [t] \prec [f([x])] \stackrel{\delta}{\Longrightarrow} [f([x])]([t]) \\ & \stackrel{(11)}{\Longrightarrow} & \exists \underline{z} \equiv t, x_1 \equiv x : f(x_1)(\underline{z}) \\ & & {\scriptstyle \exists x_1 \equiv x\ :\ t \equiv f(x_1)} \\ f(x) \prec g(x) & \stackrel{\tau}{\Longrightarrow} & [f([x])] \prec [g([x])] \stackrel{\delta}{\Longrightarrow} \forall [\underline{z}] : [f([x])]([\underline{z}]) \Rightarrow [g([x])]([\underline{z}]) \\ & \stackrel{(11)}{\Longrightarrow} & \forall \underline{z} : (\exists \underline{z}_1 \equiv \underline{z}, x_1 \equiv x : f(x_1)(\underline{z}_1)) \Rightarrow (\exists \underline{z}_2 \equiv \underline{z}, x_2 \equiv x : g(x_2)(\underline{z}_2)) \\ & \Longleftrightarrow & \forall \underline{z}, \underline{z}_1 \equiv \underline{z}, x_1 \equiv x\ \exists \underline{z}_2 \equiv \underline{z}, x_2 \equiv x : f(x_1)(\underline{z}_1) \Rightarrow g(x_2)(\underline{z}_2) \\ & & {\scriptstyle \forall x_1 \equiv x\ \exists x_2 \equiv x\ :\ f(x_1) \equiv g(x_2)} \end{array} \tag{12}$$

The above definition considers only the topmost terms of the refined sort. As indicated in the table 3, one has to introduce new variables and quantifiers for all subterms of the refined sort.

4.2.2 Semantics

We define the multialgebraic semantics of specifications following [WM95b].

Definition 4.3 Given $\Sigma = \langle \mathcal{S}, \mathcal{F} \rangle$, a Σ-multialgebra M is defined by:

- a carrier S^M for each $S \in \mathcal{S}$
- a set valued function $f^M : S_1^M \dots S_n^M \to \mathcal{P}^+(S^M)$, for each $f : S_1 \dots S_n \to S \in \mathcal{F}$.

For an assignment $\beta : X \to M$, $M \models_\beta \phi$ for ϕ:

- $s \doteq t$ iff $\beta[s] = \beta[t] = \{\alpha\}$, for some $\alpha \in |M|$,
- $s \prec t$ iff $\beta[s] \subseteq \beta[t]$,

Variables are assigned individuals – not sets. The definition implies that $\doteq$ is a *partial equivalence*: $t \doteq t$ is a tautology only when t is a variable, and is false when t is nondeterministic.

The quotient construction is defined analogously to the quotient of usual (deterministic) algebras:

Definition 4.4 Given a multialgebra M and an equivalence $\equiv$, the quotient $M/_\equiv$ is defined as:

- $|M/_\equiv| = \{[x] : x \in |M|\}$
- $c^{M/_\equiv} = \{[s] : s \in c^M\}$
- $f^{M/_\equiv}([x_1], ..., [x_n]) = \{[f^M(y_1, ..., y_n)] : y_i \in [x_i]\}$

Notice that, since $\equiv$ is not a congruence, it is not generally the case that $[f([x])] = [f(x)]$. Therefore, in the last point, we have to gather the equivalence classes $[f(y_i)]$ for all $y_i \in [x]$. As in the case of relations, one can easily verify that the mapping $x \mapsto [x]$ defines a (multi)homomorphism from A to $A/_\equiv$. In fact, relations are isomorphic to multialgebras and the following counterpart of theorem 4.2, expressing correctness of the verification criteria defined by τ in (12), should not be a surprise. We give an independent proof because the multialgebraic language is slightly different from the relational one.

Theorem 4.5 $M \models \tau(\mathcal{A}) \iff M/_\equiv \models \delta(\mathcal{A})$.

Proof. We proceed by induction on the complexity of the formula $\phi \in \mathcal{A}$ and show the theorem for the atomic (sub)formulas. Verification of the statement for the terms of non-refined sorts is obvious, and so it is for the deterministic terms. We consider only the case where all subterms are refined.

1. $\phi = f(x) \doteq t$:

   $M \models \tau(\phi) \iff M \models \forall \underline{z}, \underline{z}_1 \equiv \underline{z}, x_1 \equiv x : f(x_1)(\underline{z}_1) \Rightarrow \underline{z} \equiv t \iff$
   $M \models \forall \underline{z} : (\exists \underline{z}_1 \equiv z, x_1 \equiv x : f(x_1)(\underline{z}_1)) \Rightarrow \underline{z} \equiv t \overset{(11)}{\iff}$
   $M \models \forall \underline{z} : [f([x])([\underline{z}]) \Rightarrow [\underline{z}] \doteq [t] \iff M/_\equiv \models \forall \underline{z} : f(x)(\underline{z}) \Rightarrow \underline{z} \doteq t \iff M/_\equiv \models \delta(\phi)$

2. $\phi = t \prec f(x)$:

   $M \models \tau(\phi) \iff M \models \exists \underline{z} \equiv t, x_1 \equiv x : f(x_1)(\underline{z}) \overset{(11)}{\iff} M \models [f([x])]([t]) \iff$
   $M/_\equiv \models f(x)(t) \iff M/_\equiv \models t \prec f(x) \iff M/_\equiv \models \delta(\phi)$

3. $\phi = f(x) \prec g(x)$:

   $M \models \tau(\phi) \iff M \models \forall \underline{z}, \underline{z}_1 \equiv \underline{z}, x_1 \equiv x \ \exists \underline{z}_2 \equiv \underline{z}, x_2 \equiv x : f(x_1)(\underline{z}_1) \Rightarrow g(x_2)(\underline{z}_2) \iff$
   $M \models \forall \underline{z} : (\exists \underline{z}_1 \equiv \underline{z}, x_1 \equiv x : f(x_1)(\underline{z}_1)) \Rightarrow \exists \underline{z}_2 \equiv \underline{z}, x_2 \equiv x : g(x_2)(\underline{z}_2) \overset{(11)}{\iff}$
   $M \models \forall \underline{z} : [f([x])]([\underline{z}]) \Rightarrow [g([x])]([\underline{z}]) \iff M/_\equiv \models \forall \underline{z} : f(x)(\underline{z}) \Rightarrow g(x)(\underline{z}) \iff M/_\equiv \models \delta(\phi)$

$\square$

4.3 Predicates on Functions

A nondeterministic operation is here represented by a set of functions (see [Broy93]). A possible result of $N_f(x)$ corresponds to picking a function h which satisfies the predicate $P_f(h)$ and applying it to x.

$$
\begin{array}{rcl}
\mathbf{N}_f(s) \doteq t & \stackrel{\nu}{\Longrightarrow} & P_f(h) \Rightarrow h(s) \doteq t \\
N_f(s) \doteq t & \stackrel{\nu}{\Longrightarrow} & \exists h : P_f(h) \wedge h(s) \doteq t \\
\mathbf{N}_f(s) \doteq N_g(s) & \stackrel{\nu}{\Longrightarrow} & P_f(h) \Rightarrow \exists h_1 : P_g(h_1) \wedge h(s) \doteq h_1(s)
\end{array}
\tag{13}
$$

Notice that, unlike in the relational or multialgebraic case, the last formula does not correspond to an implication between the respective predicates $P_f(h) \Rightarrow P_g(h)$.

The specification looks now as follows:

$$
\begin{array}{lrcl}
1. & (P_\sqcup(h) \Rightarrow h(S) \in S) & \vee & S \doteq \emptyset \\
2. & P_{dfs}(h) & \Rightarrow & \exists h_1 : P_{tr}(h_1) \wedge h(v) \doteq h_1(v, ch(v), \emptyset) \\
3. & P_{tr}(h) & \Rightarrow & h(v, \emptyset, T) \doteq T \\
4. & S \not\doteq \emptyset \wedge (\exists h : P_\sqcup(h) \wedge x \doteq h(S)) \wedge x \sqsubset T & \Rightarrow & P_{tr}(h) \Rightarrow \exists h_1 : P_{tr}(h_1) \wedge \\
 & & & \qquad h(v, S, T) \doteq h_1(v, S \setminus x, T) \\
5. & S \not\doteq \emptyset \wedge (\exists h : P_\sqcup(h) \wedge x \doteq h(S)) \wedge x \not\sqsubset T & \Rightarrow & P_{tr}(h) \Rightarrow \exists h_1, h_2 : P_{tr}(h_1) \wedge P_{tr}(h_2) \wedge \\
 & & & h(v, S, T) \doteq h_1(v, S \setminus x, h_2(x, ch(x), T \circ \langle v, x \rangle))
\end{array}
$$

As for relations, we can give an equivalent formulation collecting the universal occurrences:

$$
\begin{array}{llll}
3. & P_{tr}(h) & \Rightarrow & h(v, \emptyset, T) \doteq T \\
4. & & \wedge & S \not\doteq \emptyset \wedge (\exists h : P_\sqcup(h) \wedge x \doteq h(S)) \wedge x \sqsubset T \Rightarrow \exists h_1 : P_{tr}(h_1) \wedge \\
 & & & \qquad h(v, S, T) \doteq h_1(v, S \setminus x, T) \\
5. & & \wedge & S \not\doteq \emptyset \wedge (\exists h : P_\sqcup(h) \wedge x \doteq h(S)) \wedge x \not\sqsubset T \Rightarrow \exists h_1, h_2 : P_{tr}(h_1) \wedge P_{tr}(h_2) \wedge \\
 & & & h(v, S, T) \doteq h_1(v, S \setminus x, h_2(x, ch(x), T \circ \langle v, x \rangle))
\end{array}
$$

This is a translation in conventional higher order logic.

4.3.1 The Proof Obligations for Data Refinement

As in the case of relations, the translation ν yields formulas in a standard (higher order) language and we may take $\delta = id$. Following (6), the refinement translation τ is obtained simply by adding respective quantifier over the (refined) argument:

$$
\begin{array}{rcl}
P_f(h) \Rightarrow h(s) \doteq t & \stackrel{\tau}{\Longrightarrow} & \forall s_1 \equiv s : P_f(h) \Rightarrow h(s_1) \equiv t \\
\exists h : P_f(h) \wedge h(s) \doteq t & \stackrel{\tau}{\Longrightarrow} & \exists s_1 \equiv s \; \exists h : P_f(h) \wedge h(s_1) \equiv t \\
P_f(h) \Rightarrow \exists h_1 : P_g(h_1) \wedge h(s) \doteq h_1(s) & \stackrel{\tau}{\Longrightarrow} & P_f(h) \Rightarrow \forall s_1 \equiv s \; \exists s_2 \equiv s \; \exists h_1 : \\
 & & \qquad P_g(h_1) \wedge h(s_1) \equiv h_1(s_2)
\end{array}
\tag{14}
$$

If the predicate $P_f(h)$ determines a unique function f, we may remove all the occurrences of $P_f(h)$ and quantification over h, and replace the occurrences of h by f. The result will be then the same as in the case of relations (second line in (8)).

4.3.2 Semantics

The semantics is straightforward by associating a set of functions with each nondeterministic function symbol.

Definition 4.6 Given $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{N} \rangle$, a multifunctional Σ-structure A is defined by:

- a carrier S^A for each $S \in \mathcal{S}$
- a function $f^A : S^A \to T^A$ for each $f : S \to T \; \in \mathcal{F}$
- a set of functions $P_f^A = \bigcup_i f_i : S^A \to T^A$ for each $P_f : (S \to T) \to Bool \; \in \mathcal{N}$.

The satisfaction of axioms, in particular those of the form (13), is defined in the standard way.

By analogy with (6) and the two previous examples, the translation (14) should appear quite plausible. However, in the current semantic apparatus there seems to be no natural construction corresponding to the quotient. We may take the quotient on the underlying domain of individuals but not over sets of functions interpreting nondeterministic operations. Using the more abstract (relational or multialgebraic) view, which considered only input-output relation, we could start with the function *fst*, a sequence S and its equivalence class $[S]$, and map $fst(S) \mapsto fst([S])$. This image defined then the input-output relation in the quotient structure. But having a (set of) function(s) F', we cannot, in general, map it on another set F of which F' should be an implementation. The simplest reason is that, when an implementation is more deterministic than the abstract specification, the set F will contain more elements than F'.

An apparent solution would be not to define any corresponding semantic construction. Having a set of axioms $\nu(\mathcal{A})$ and its model class *Mod*, one could say that the model class *Mod'* of the axiom set $\tau(\mathcal{A})$ is *defined* to be an implementation of *Mod*. In this case the correctness of the verification condition is trivial. However, we lose the basic notion of implementation as the subset relation on the model classes. Not only $Mod' \not\subseteq Mod$, but there is no construction κ such that $\kappa(Mod') \subseteq Mod$. Thus, for instance, transitivity of the implementation relation has to be proved in a much more cumbersome way than in the presence of such a simple semantic definition.

To cope with this problem, we will "simulate" the abstract quotient constructions as defined for relations and multialgebras by constructing, given a (set of) concrete function(s) F', an appropriate set of abstract functions F which satisfy the defining predicate and model the intended input-output relation. F is only one among the possible realizations of the relation determined by F'. However, showing that it satisfies the abstract axioms, provided that F' satisfies their τ-translation, we ensure correctness of the implementation. Notice that this does not correspond to any notion of homomorphism with respect to the functions contained in the respective sets F', F. The natural definition of a homomorphism could be only derived from the multialgebra homomorphism by the requirement that $h : F' \mapsto F$ and that the result set $h(F'(x)) \subseteq F(h(x))$.

Definition 4.7 Given a (multifunctional) Σ-structure A and an equivalence relation $\equiv$, the *quotient structure* $A/_{\equiv}$ is defined as follows:

- $|A/_{\equiv}| = \{[x] : x \in |A|\}$
- $c^{A/_{\equiv}} = [c]$ for each $c \in \mathcal{F}$
- $f^{A/_{\equiv}}([x_i]) = [f^A(x_i)]$ for each $f \in \mathcal{F}$
- for each $P : (S \to T) \to Bool \in \mathcal{N}$, we have a set of functions $P^A = \bigcup_i p_i : S^A \to T^A$. Let $p_i([x]) = \bigcup_{y \in [x]} p_i(y)$ and $p([x]) = \bigcup_i p_i([x])$. This last set may be seen as a set of pairs $\bigcup_{i, y \in [x]} \langle y, p_i(y) \rangle$. Let $Pp = \prod_{x \in S} p([x])$ be the cartesian product of $p([x])$ over all $x \in S^A$. Each element of Pp has the form $\langle \langle y_1, p_{i1}(y_1) \rangle, \langle y_2, p_{i2}(y_2) \rangle, \langle y_3, p_{i3}(y_3) \rangle ... \rangle$ where $m \neq n \Rightarrow y_m \not\equiv y_n$. Thus it defines a function from $S^{A/_{\equiv}}$ to $T^{A/_{\equiv}}$ and we let $P^{A/_{\equiv}} = Pp$.

Theorem 4.8 $A \models \tau(\mathcal{A}) \iff A/_{\equiv} \models \mathcal{A}$.

Proof. If we forget the predicate part, the quotient construction and verification for the functions are the same as in the deterministic case. We show the theorem for the atomic formulas ϕ affected by τ. The following two facts are obvious from the construction in definition 4.7:

$$\begin{array}{rcl} A \models \exists s_1 \equiv s\ \exists h : P(h) \land h(s_1) \equiv t & \iff & A/_{\equiv} \models \exists h : P(h) \land h(s) \doteq t \\ A \models \forall s_1 \equiv s : P(h) \Rightarrow h(s_1) \equiv t & \iff & A/_{\equiv} \models P(h) \Rightarrow h(s) \doteq t \end{array} \tag{15}$$

1. Let $\phi = P_f(h) \Rightarrow h(s) \doteq t$:
   $A \models \tau(\phi) \iff A \models \forall s_1 \equiv s : P_f(h) \Rightarrow h(s_1) \equiv t \overset{(15)}{\iff} A/_{\equiv} \models P_f(h) \Rightarrow h(s) \doteq t \iff A/_{\equiv} \models \phi$
2. Let $\phi = \exists h : P_f(h) \land h(s) \doteq t$:
   $A \models \tau(\phi) \iff A \models \exists s_1 \equiv s\ \exists h : P_f(h) \land h(s_1) \equiv t \overset{(15)}{\iff} A/_{\equiv} \models \exists h : P_f(h) \land h(s) \doteq t \iff$ $A/_{\equiv} \models \phi$

3. Let $\phi = P_f(h) \Rightarrow \exists h_1 : P_g(h_1) \wedge h(s) \doteq h_1(s)$:
$A \models \tau(\phi) \iff A \models P_f(h) \Rightarrow \forall s_1 \equiv s\ \exists s_2 \equiv s\ \exists h_1 : P_g(h_1) \wedge h(s_1) \equiv h_1(s_2) \overset{(15)}{\iff}$
$A/_{\equiv} \models P_f(h) \Rightarrow \exists h_1 : P_g(h_1) \wedge h(s) \doteq h_1(s) \iff A/_{\equiv} \models \phi$

□

Remark. The equivalences (15) hold only for the specific kinds of formulas. For instance, the fact (*): $SEQ(E) \models P_{fst}(h) \Rightarrow h(\epsilon\hat{\ }x\hat{\ }y) \doteq x$ does not mean that $SEQ(E)/_{\equiv} \models P_{fst}(h) \Rightarrow h([\epsilon\hat{\ }x\hat{\ }y]) \doteq x$, where $\equiv$ is as defined in the implementation of $SET(E)$ (axioms $A1', A2', U3', E0 - E3$). However, (*) implies $SEQ(E) \models \exists s \equiv \epsilon\hat{\ }x\hat{\ }y\ \exists h : P_{fst}(h) \wedge h(s) \doteq x$, from which we can conclude that $SEQ(E)/_{\equiv} \models \exists h : P_{fst}(h) \wedge h([\epsilon\hat{\ }x\hat{\ }y]) \doteq x$.

4.4 Oracles

In the context of algebraic specifications, the oracle semantics of nondeterminism is studied, for instance, in [Wal93], [WM95a]. It models a nondeterministic operation by a function with additional parameter called *oracle* or *index*. A possible result of $N_f(x)$ corresponds to picking a particular value i of the oracle argument and evaluating $f(i, x)$.

$$\begin{array}{rcl} \mathbf{N}_f(s) \doteq t & \overset{\nu}{\Longrightarrow} & \forall i : f(i,s) \doteq t \\ N_f(s) \doteq t & \overset{\nu}{\Longrightarrow} & \exists i : f(i,s) \doteq t \\ \mathbf{N}_f(s) \doteq N_g(s) & \overset{\nu}{\Longrightarrow} & \forall i\ \exists j : f(i,s) \doteq g(j,s) \end{array} \tag{16}$$

The specification looks now as follows:

$$\begin{array}{lrcl} 1. & \forall i : \sqcup(i,S) \in S & \vee & S \doteq \emptyset \\ 2. & \forall i\ \exists j & : & dfs(i,v) \doteq tr(j,v,ch(v),\emptyset) \\ 3. & \forall i : tr(i,v,\emptyset,T) & \doteq & T \\ 4. & S \not\doteq \emptyset \wedge (\exists i : x \doteq \sqcup(i,S)) \wedge x \sqsubset T & \Rightarrow & \forall i\ \exists j : tr(i,v,S,T) \doteq tr(j,v,S \setminus x,T) \\ 5. & S \not\doteq \emptyset \wedge (\exists i : x \doteq \sqcup(i,S)) \wedge x \not\sqsubset T & \Rightarrow & \forall i\ \exists j,k : tr(i,v,S,T) \doteq tr(j,v,S \setminus x, \\ & & & \qquad tr(k,x,ch(x),T \circ \langle v,x \rangle)) \end{array}$$

Note that the model using sets of functions is very close to the idea of oracles.

4.4.1 The Proof Obligations for Data Refinement

Again, the translation ν yields formulas in a standard language and we may take $\delta = id$. Following (6), the refinement translation τ is obtained simply by adding respective quantifier over the (refined) argument:

$$\begin{array}{rcl} \forall i : f(i,s) \doteq t & \overset{\tau}{\Longrightarrow} & \forall i\ \forall s_1 \equiv s : f(i,s) \doteq t \\ \exists i : f(i,s) \doteq t & \overset{\tau}{\Longrightarrow} & \exists i\ \exists s_1 \equiv s : f(i,s_1) \doteq t \\ \forall i\ \exists j : f(i,s) \doteq g(j,s) & \overset{\tau}{\Longrightarrow} & \forall i\ \forall s_1 \equiv s\ \exists j\ \exists s_2 \equiv s : f(i,s_1) \doteq g(j,s_2) \end{array} \tag{17}$$

In contrast to the model using sets of functions with oracles we do not use higher order logic.

4.4.2 Semantics

A similar problem with the semantic counterpart of the logical refinement notion as indicated in subsection 4.3.2 will occur here. If a nondeterministic operation is implemented by a function – which therefore does not depend on the oracle parameter – the standard quotient construction would not work. If we attempt to gather the results of the function on all values equivalent to the given one in order to obtain a nondeterministic operation, then there is no direct way to express this new operation as an oracle-dependent function, since one has to find the correlation between the (new) oracle values and the required results. We could attempt to give a new quotient construction analogously to the construction in subsection 4.3.2 but now we would have to construct not only a new function but also a new oracle sort – since the concrete structure may have only one element of the oracle sort, if all its operations are deterministic, and we have to produce at least as many oracle elements in the abstract structure as the cardinality of “the most nondeterministic set”. For the same reason, even if we manage such a construction, it certainly would not correspond to any homomorphism.

Therefore, in the following, we will focus on the three other approaches but not on the oracles.

5 Implementation of Nondeterministic Data Types

We have considered data refinement as a special – and particularly problematic – case of implementation of nondeterministic data types. Now, we will consider a couple of other specification-building operators and the general notion of implementation. Unlike quotient, these concepts generalize easily to the nondeterministic context.

Given a specification SP, its model class (however it is defined), is denoted $Mod(SP)$. We write $SP \rightsquigarrow SP'$ if $Mod(SP') \subseteq Mod(SP)$, and consider the following specification-building operations:

$$\begin{array}{lll}
(SP/_{\equiv}) & \textbf{quotient } SP \textbf{ by} \equiv : & Spec(\Sigma) \to Spec(\Sigma) \ where \\
 & & Mod(SP') = \{M/_{\equiv} : M \in Mod(SP)\} \\
(SP|_{\sigma}) & \textbf{derive } SP \textbf{ by } \sigma : & Spec(\Sigma) \to Spec(\Sigma') \ where \\
 & & \sigma : \Sigma' \to \Sigma \\
 & & Mod(SP') = \{M|_{\sigma} : M \in Mod(SP)\} \\
\multicolumn{2}{l}{\textbf{enrich } SP \textbf{ by } \mathcal{S}', \mathcal{F}', \mathcal{N}', \mathcal{A}' :} & Spec(\Sigma) \to Spec(\Sigma') \ where \\
 & & Sig(\textbf{enrich } SP \textbf{ by } \mathcal{S}', \mathcal{F}', \mathcal{N}', \mathcal{A}') = \Sigma' \\
 & & Mod(\textbf{enrich } SP \textbf{ by } \mathcal{S}', \mathcal{F}', \mathcal{N}', \mathcal{A}') = Mod(\langle \Sigma', \mathcal{A} \cup \mathcal{A}' \rangle)
\end{array}$$

Without loss of generality, we can assume that, in the quotient construction, the signature morphism σ is an identity except for the equality symbol, for which $\sigma(\doteq) = \equiv$. Composing **quotient** with **derive** yields the appropriate relation.

Generally, in **derive**, σ is a standard signature morphism which does not (necessarily) distinguish between the $\mathcal{N}$ and $\mathcal{F}$ operations, that is, it maps $\mathcal{N}' \cup \mathcal{F}'$ into $\mathcal{N} \cup \mathcal{F}$, under the usual restrictions on signature morphisms. However, since the operations in $\mathcal{F}$ respect the congruence while those in $\mathcal{N}$ do not, it is natural to restrict immediately the class of the relevant morphisms to such σ's that $\sigma(f) \in \mathcal{F} \Rightarrow f \in \mathcal{F}'$.

The use of notation in **enrich** is subject to the following restrictions: $SP = \langle \Sigma, \mathcal{A} \rangle$, where $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{N} \rangle$, $\mathcal{S}' \cap \mathcal{S} = \emptyset$ is a new set of sorts, $(\mathcal{F}' \cup \mathcal{N}') \cap (\mathcal{F} \cup \mathcal{N}) = \emptyset$ a set of new operation symbols sorted over $\mathcal{S} \cup \mathcal{S}'$, $\Sigma' = \Sigma \cup \langle \mathcal{S}', \mathcal{F}', \mathcal{N}' \rangle$, and $\mathcal{A}'$ is a set of axioms over Σ'.

We do not introduce an operator returning reachable restrictions of models. Under the assumption from the introduction that all constructors are deterministic, such an operator may be defined in exactly the same way as for the deterministic specifications: **reachable** SP **on** S **with** T, assumes that T are deterministic terms. A generalization admitting also nondeterministic terms in T is not straightforward, since it has to consider various possible notions of reachability with such terms. This issue is discussed in [WM95c].

Following [ST88], we say that a specification-building operation κ is a *constructor* if it is defined as an operation on models, i.e., if $Mod(\kappa(SP)) = \{\kappa(A) : A \in Mod(SP)\}$. All three cases of the quotient construction which we have given (definitions 4.1, 4.4, 4.7), yield a constructor. The standard definition of reduct in our case makes it a constructor, too. We consider *constructor implementations* and write $SP \overset{\kappa}{\rightsquigarrow} SP'$ if $SP \rightsquigarrow \kappa(SP')$. For instance:

$$SP \overset{\sigma}{\rightsquigarrow} SP' \ if \ SP \rightsquigarrow SP'|_{\sigma} \quad SP \overset{\equiv}{\rightsquigarrow} SP' \ if \ SP \rightsquigarrow SP'/_{\equiv} \quad SP \overset{\sigma;\equiv}{\rightsquigarrow} SP' \ if \ SP \overset{\sigma}{\rightsquigarrow} SP'/_{\equiv}$$

The proof of the following vertical composition lemma is trivial:

Lemma 5.1 . *(1)* **quotient**, **derive** *and* **enrich** *are monotonic.*
(2) $\overset{*}{\leadsto}$ *is transitive – for constructors* α, β *we have:*
$SP \overset{\alpha}{\leadsto} SP' \wedge SP' \overset{\beta}{\leadsto} SP'' \Rightarrow SP \overset{\alpha;\beta}{\leadsto} SP''$.

Proof. (1) for **enrich** is trivial. Since $\_/_{\equiv}$ and $\_|\sigma$ are constructors, the rest of (1) and (2) is just theorem 4.14 from [ST88]. □

5.1 Verification of the Implementation Relation

As indicated before, in order to verify the specifications we translate the axioms into some standard logic. Typically, the translation will yield rather complicated formulas, but we have chosen this approach because verification can now utilize the (semi-)automatic tools designed for proofs in a standard logical setting.

Let $SP = \langle \Sigma, \mathcal{A} \rangle, SP' = \langle \Sigma', \mathcal{A}' \rangle$ be the implemented and the implementing specifications, respectively. To verify the implementation we have to show for

$$\begin{array}{rcl} SP \leadsto SP' & : & \mathcal{A}' \vdash \mathcal{A} \\ & & \delta(\nu(\mathcal{A}')) \vdash \delta(\nu(\mathcal{A})) \\ SP \overset{\sigma}{\leadsto} SP' & : & \mathcal{A}' \vdash \sigma(\mathcal{A}) \\ & & \delta(\nu(\mathcal{A}')) \vdash \delta(\nu(\sigma(\mathcal{A}))) \\ SP \overset{\equiv}{\leadsto} SP' & : & \mathcal{A}' \vdash \tau(\mathcal{A}) \\ & & \delta(\nu(\mathcal{A}')) \vdash \tau(\nu(\mathcal{A})) \end{array} \tag{18}$$

In the first two cases, verification is exactly as for the deterministic specifications. (If specifications are written in the abstract language of section 3, we apply the translation schemata for the intended semantics.) The last case has been studied in the preceding sections.

We illustrate this verifying that $SET \overset{\sigma,\equiv}{\leadsto} SEQ$, where the respective specifications and σ, $\equiv$ are given in section 2. We let SET, SEQ stand for the ν-translated versions of the respective specifications as they have been given in the earlier sections. For SEQ, we have $\nu(SEQ) = SEQ$, since it is deterministic. For the implementation step, we have to show $SEQ/_{\equiv} \models \sigma(SET)$, and the whole proof obligation is to show $\delta(SEQ) \vdash \tau(\sigma(SET))$. For relations and sets of functions we have $\delta = id$. For multialgebras, since SEQ is deterministic and contains only equations, the result will be $\delta(SEQ) = SEQ$. Thus, in all three cases, we have to show that $SEQ \vdash \tau(\sigma(SET))$.

We list the respective SET axioms for choice and their $\tau \circ \sigma$-translation for each of the three considered cases. Since *fst* is a function, we immediately simplify the translated axioms as indicated in the respective definitions of τ:

$$\begin{array}{lcl} \forall S : (\forall x : R_{\sqcup}(S,x) \Rightarrow x \in S) \vee S \doteq \emptyset & \overset{\sigma}{\Longrightarrow} & \forall S : (\forall x : R_{fst}(S,x) \Rightarrow x \in S) \vee S \equiv \epsilon \\ & \overset{(8)}{\Longrightarrow} & \forall S\ \forall S_1 \equiv S : fst(S_1) \in S \vee S \equiv \epsilon \\ \forall S : (P_{\sqcup}(h) \Rightarrow h(S) \in S) \vee S \doteq \emptyset & \overset{\sigma}{\Longrightarrow} & \forall S : (P_{fst}(h) \Rightarrow h(S) \in S) \vee S \equiv \epsilon \\ & \overset{(14)}{\Longrightarrow} & \forall S\ \forall S_1 \equiv S : fst(S_1) \in S \vee S \equiv \epsilon \\ \forall S : \sqcup(S) \in S \vee S \doteq \emptyset & \overset{\sigma}{\Longrightarrow} & \forall S : fst(S) \in S \vee S \equiv \epsilon \\ & \overset{(12)}{\Longrightarrow} & \forall S\ \forall S_1 \equiv S : fst(S_1) \in S \vee S \equiv \epsilon \end{array}$$

Incidentally, due to the determinacy of the whole SEQ, and *fst* in particular, all the cases yield identical proof obligations. Verification of the fact $SEQ \vdash \forall S\ \forall S_1 \equiv S : fst(S_1) \in S \vee S \equiv \epsilon$ is an easy exercise.

6 Parameterisation

One of the central tenets of specification-in-large is the modularity of specifications and verification. Having identified two component specifications $SP1$ and $SP2$ which are composed using a specification-building operator R into a specification $R(SP1, SP2)$, we are interested in the possibility of refining the two components independently from each other, in such a way that the resulting specifications $SP1'$ and $SP2'$ can be composed using R and yield $R(SP1', SP2')$ which implements $R(SP1, SP2)$. For instance, if $SP1$ is some basic specification which is used by $SP2$, we would like to implement $SP1$ and be able to use this implementation in $SP2$ instead of the original $SP1$. The general framework should guarantee that the result $SP2$ *using* $SP1'$ is a correct implementation of $SP2$ *using* $SP1$. Similarly, we would like an implementation $SP2'$ of $SP2$ to yield an implementation $SP2'$ *using* $SP1'$ of $SP2$ *using* $SP1'$. The results of this section allow us to do this.

The issue of modularity is captured by the classical notion of parameterised specifications. We consider it as a just another specification-building operation with the semantics defined not as a function (or push-out) but along the same lines as, for instance, the semantics of **enrich** or **derive**. It corresponds to the fact that one specification may "use" another one yielding a hierarchical structure of the specification components. The aspect of "parameterisation" is reflected in the definition of implementation which takes into account the structure of the specification, and in the restrictions (*conservativity*, subsection 6.1) on the form of such specifications, which correspond to the standard requirements of protecting the used component (actual parameter protection). The first part of this section does not rely on such restrictions and is merely an adaptation of basic notions and results from [ST88].

Again, we consider relations, multialgebras, and predicates on functions. The following definitions and lemmas apply uniformly without further generalization to all three cases.

Definition 6.1 Given specifications $SP_p \subseteq SP_r$ over signatures $\Sigma_p \subseteq \Sigma_r$, $\lambda X : SP_p.SP_r(X)$ is a parameterised specification with the formal parameter SP_p,

- $Sig(\lambda X : SP_p.SP_r(X)) = \Sigma_r$
- $Mod(\lambda X : SP_p.SP_r(X)) = Mod(SP_r({}^{SP_p}/_X))$

Notice that the second point defines the semantics of a parameterised specification as the semantics of a non-parameterised specification. The λ-notation is a mere abbreviation for **enrich** SP_p **by** P where $P = SP_r \setminus SP_p$, which is applied to indicate that the implementation relation is defined differently (below). Notice that the formal parameter is a specification. The actual parameter is required to satisfy this specification. We say that SP is a *legal* (actual) parameter iff $SP_p \rightsquigarrow SP$.[4]

Let $SET(E)$ be the specification from section 2 (Figure 1, with the modified choice signature and axiom) viewed as a parameterised specification $\lambda X : \langle E, \{\doteq\}, \emptyset\rangle \,.\, SET(X)$. By definition, its parameter must have the signature $\Sigma_p = \langle E, \{\doteq\}\rangle$, and the resulting specification has the signature $\Sigma_r = \langle\{E, Set(E)\}, \{\emptyset, \circ, \cup, \in, \sqcup, \setminus, \doteq\}\rangle$. To instantiate with $NAT = \langle Nat, \{0, s, +, \doteq\}, \emptyset\rangle$, we need a "fitting" morphism $\sigma : E \to NAT$ and the signature of the resulting specification does not contain $0, s, +$! As a matter of fact, this parameter passing does not yield $SET(NAT)$, but $SET(NAT|_\sigma)$. The other way around, it is illegal to pass NAT as a parameter since it does not have the signature Σ_p.

However, before passing NAT (or after passing $NAT|_\sigma$) we may rename the sort E to Nat. More generally, we need to show the following fact: if $E \overset{\sigma}{\rightsquigarrow} NAT$ then $SET(E) \underset{\sigma}{\circ\!\!\rightsquigarrow} SET(NAT)$, where the latter relation is written so to emphasize that it concerns implementation of the parameter in a parameterised specification – the notation $P(E) \underset{\kappa}{\circ\!\!\rightsquigarrow} P(E')$ stands for $P(E) \circ\!\!\rightsquigarrow P(\kappa(E'))$ and $\circ\!\!\rightsquigarrow$ is defined below.[5]

[4] The "fitting" morphism is then not just a signature morphism but a specification morphism $\sigma : \langle\Sigma_p, \mathcal{A}_p\rangle \to \langle\Sigma, \mathcal{A}\rangle$, where, for each $A \in \mathcal{A}_p : \mathcal{A} \models \sigma(A)$. Since we are considering only the case where the formal and the actual parameters have the same signature, we only require that $\forall A \in \mathcal{A}_p : \mathcal{A} \models A$.

[5] To be quite precise, we would have to identify which parameter is concerned (if there are more than one), and what fitting morphism is used. We refrain from cluttering notation in this way since it does not seem necessary for our example.

$$\begin{tikzcd} E \arrow[r, hook] \arrow[d, "\overset{\sigma}{\rightsquigarrow}"] & SET(E) \arrow[d, "\underset{\sigma}{\circ\!\!\!\rightsquigarrow}"] \\ NAT \arrow[r] & SET(NAT) \end{tikzcd}$$

The signature of $SET(NAT|_\sigma)$ is still Σ_r. If $SP_p \overset{*}{\rightsquigarrow} SP_a$, the notation $SP_r(SP_a)$ implicitly indicates an appropriate renaming of formal parameter, as for $Sig(SET(NAT)) = \langle\{Nat, Set(Nat)\}, \{...\}\rangle$. The lower arrow in the diagram represents the corresponding partial signature morphism.

Implementation of one parameterised specification by another is defined pointwise:

Definition 6.2 Let P, P' be parameterised specifications over the same parameter specification SP_p:

- $P \circ\!\!\!\rightsquigarrow P'$ iff, for all legal SP_p parameters SP : $P(SP) \rightsquigarrow P'(SP)$;
- for $P = \lambda X : SP_p.SP_r(X)$, and a specification building operator $\kappa : Spec(\Sigma_r) \to Spec(\Sigma)$, $\kappa(P)$ is defined as $\lambda X : SP_p.\kappa(SP_r(X))$;
- $P \overset{\kappa}{\circ\!\!\!\rightsquigarrow} P'$ iff $P \circ\!\!\!\rightsquigarrow \kappa(P')$.

Since there is no difference between passing an actual parameter and implementing the formal one, and the latter yields a parameterised specification, we will write the parameterised specifications simply as $SP_r(SP_p)$.

According to the definition, Σ_p remains unchanged, but since $\Sigma_p \subseteq \Sigma_r$, it does not imply that implementation of a parameterised specification affects only the body, but not the parameter part. Such a restriction will be imposed by the the requirement of *conservativity* discussed in the following subsection.

The definition 6.2 introduces an implementation relation and it may seem that, since this relation is defined pointwise with reference to the implementation of non-parameterised specifications, it is really a new relation. The following lemma shows that the two actually coincide and for this reason we can dispense with the additional symbol $\circ\!\!\!\rightsquigarrow$.

Lemma 6.3 *Let $P = SP_r(SP_p)$ and $P' = SP_r'(SP_p)$ be two parameterised specifications. Then $P \overset{\kappa}{\rightsquigarrow} P' \Rightarrow P \overset{\kappa}{\circ\!\!\!\rightsquigarrow} P'$.*

Proof. Consider first the case $\kappa = id$. Let $SP_p \rightsquigarrow SP$ be an arbitrary legal parameter. By lemma 5.1, $Mod(SP_r'(SP)) \subseteq Mod(SP_r(SP_p))$. So, $\forall M \in Mod(SP_r'(SP)) : M \models SP_r(SP_p) \cup SP$. We have that $Mod(SP_r(SP_p) \cup SP) = Mod(SP_r \cup SP_p \cup SP) = Mod(SP_r \cup SP) = Mod(SP_r(SP))$, which implies $M \in Mod(SP_r(SP))$. The same argument with SP_r' replaced by $\kappa(SP_r')$ shows the general statement. $\square$

The following lemma makes the required diagram above commute:

Lemma 6.4 *For a parameterised specification P:*

1. *if $SP \rightsquigarrow SP'$ then $SP_r(SP) \rightsquigarrow SP_r(SP')$,*
2. *if $SP \overset{\kappa}{\rightsquigarrow} SP'$ then $SP_r(SP) \underset{\kappa}{\rightsquigarrow} SP_r(SP')$,*

Proof. 1 is proved by a trivial induction on SP_r (instantiated $SP_r(SP)$ is **enrich** SP **by** SP_r, and specification-building operations are monotonic). 2 follows then, since $SP \overset{\kappa}{\rightsquigarrow} SP'$ means $SP \rightsquigarrow \kappa(SP')$. $\square$

This lemma applies in general to nested parameterisation and several parameters. For instance, given a parameterised specification $S1(S2(S3))$, we have $S3 \overset{\kappa}{\rightsquigarrow} S3' \overset{2.}{\Longrightarrow} S2(S3) \underset{\kappa}{\rightsquigarrow} S2(S3') \overset{def}{\Longleftrightarrow} S2(S3) \rightsquigarrow S2(\kappa(S3')) \overset{1.}{\Longrightarrow} S1(S2(S3)) \rightsquigarrow S1(S2(\kappa(S3')))$, which, abusing notation a little bit, we also write as $S1(S2(S3)) \underset{\kappa}{\rightsquigarrow} S1(S2(S3'))$.

Transitivity of the implementation relation for parameterised specifications is obtained by a pointwise application of the vertical composition of non-parameterised specifications from lemma 5.1.

Lemma 6.5 *For parameterised specifications P, P', P'' over the same parameter: if $P \overset{\alpha}{\leadsto} P'$ and $P' \overset{\beta}{\leadsto} P''$ then $P \overset{\alpha;\beta}{\leadsto} P''$.*

The main result concerning horizontal composition follows now easily.

Theorem 6.6 *If $P \overset{\kappa}{\leadsto} P'$ and $SP \overset{\mu}{\leadsto} SP'$, then $P(SP) \underset{\mu}{\overset{\kappa}{\leadsto}} P'(SP')$.*

Proof.

$$\left.\begin{array}{r} P \overset{\kappa}{\leadsto} P' \overset{D.6.2}{\Longrightarrow} P(SP) \overset{\kappa}{\leadsto} P'(SP) \\ SP \leadsto \mu(SP') \overset{L.6.4}{\Longrightarrow} P'(SP) \leadsto P'(\mu(SP')) \end{array}\right\} \overset{L.6.5}{\Longrightarrow} P(SP) \overset{\kappa}{\leadsto} P'(\mu(SP')).$$

□

As for lemma 6.4, this theorem generalizes trivially to the case of several and nested parameters.

Actually, this is not horizontal composition as formulated in [GB80] but, as argued in [ST88], it is perfectly satisfactory. It allows us to do a series of refinement steps on the parameter specification $SP \overset{\mu_1}{\leadsto} SP_1 \overset{\mu_2}{\leadsto} \ldots \overset{\mu_n}{\leadsto} SP_n$, and, independently, on the parameterised one $P \overset{\kappa_1}{\leadsto} P_1 \overset{\kappa_2}{\leadsto} \ldots \overset{\kappa_m}{\leadsto} P_m$, yielding $P(SP) \overset{\kappa_1 \ldots \kappa_m}{\leadsto} P_m(\mu_1 \ldots \mu_n(SP_n))$, or $P(SP) \underset{\mu_1 \ldots \mu_n}{\overset{\kappa_1 \ldots \kappa_m}{\leadsto}} P_m(SP_n)$.

Example 6.7 Consider the following specifications $CH, CH1$, and the parameterised specification $M(X)$, where:

$$\begin{array}{llllll} \textsf{CH} \textbf{ is} & \mathcal{S}: & D & & & \\ & \mathcal{N}: & \sqcup: & D \times D & \rightarrow & D \\ & \mathcal{A}: & C: & z \doteq N_{\sqcup}(x,y) & \Rightarrow & z \doteq x \vee z \doteq y \end{array}$$

$$\begin{array}{llllll} \textsf{CH1} \textbf{ is} & \mathcal{S}: & \ldots & & & \\ & \mathcal{N}: & \ldots & & & \\ & \mathcal{A}: & C1: & \mathbf{N}_{\sqcup}(x,y) & \doteq & x \end{array}$$

$$\begin{array}{llllll} \textsf{M} \textbf{ is } \lambda X: & \mathcal{S}_p: & D & & & \\ & \mathcal{N}_p: & \sqcup: & D \times D & \rightarrow & D \\ \textbf{enrich } X \textbf{ by} & \mathcal{N}: & m: & D \times D & \rightarrow & D \\ & \mathcal{A}: & m1: & \mathbf{N}_m(x,y) & \doteq & N_{\sqcup}(x,y) \\ & & m2: & x & \doteq & N_m(x,y) \\ & & m3: & y & \doteq & N_m(x,y) \end{array}$$

Obviously, $CH \leadsto CH1$ and hence, by theorem 6.6, $M(CH) \leadsto M(CH1)$. However, the axioms of $M(CH1)$ collapse the sort D. $m3$ and $m1$ yield $y \doteq N_{\sqcup}(x,y)$, which combined with $C1$ gives $y \doteq x$.

□

CH admits both nondeterministic and deterministic implementations (like $CH1$) while $M(X)$ requires $m(x,y)$ to return both its arguments, i.e., to be nondeterministic. Thus narrowing the range of nondeterminism in the implementation of the parameter is incompatible with the requirement of full nondeterminism in the body of the parameterised specification.

However, it is easy to see that the problem has nothing to do with (the admissible) nondeterminism of $\sqcup$ in CH. If we required in CH full nondeterminism, $CH1$ would not be a correct implementation. The problem is exactly the same as in the case of classical deterministic specifications where parameterisation may be non-persistent. Exactly the same requirement as in the standard case is needed here.

Since we adapt loose semantics, parameterisation is not a constructor. The parameter part may be extracted from a $PA \in \mathit{Mod}(P(SP))$ by $\mathit{par}(PA) = PA|_{\Sigma_p}$. And, obviously, for a given A, there may be several PA such that $\mathit{par}(PA) = A$. Writing $[\![SP_r(SP_p)]\!]$ for $\mathit{Mod}(SP_r(SP_p))$, passing an algebra A as an actual parameter yields:

$$[\![SP_r(SP_r)]\!](A) = \{PA \in \mathit{Mod}(SP_r(SP_p)) : PA|_{\Sigma_p} \simeq A\} \tag{19}$$

Viewing parameter passing in this way, automatically ensures *actual parameter protection.* However, this happens only because it excludes some SP models as actual parameters when $P(SP)$ is not persistent. In the example 6.7, only $CH1$ models with 1-element carrier would be possible parameters since only such ones can be recovered as reducts of $M(CH1)$ models. The usual persistence condition amounts in this setting to admitting *all* models of SP_p as legal parameters, i.e.,

$$\forall A \in \mathit{Mod}(SP_p) : [\![SP_r(SP_p)]\!](A) \neq \emptyset \tag{20}$$

The advantage of this definition of parameterised specifications is that it essentially reduces their implementation to implementation of non-parameterised specifications. One should observe, however, that we are using definition 6.2 – an arbitrary specification P with $Mod(P) \subseteq Mod(SP_r(SP_p))$ would not necessarily be considered as an implementation.

The definition makes also *passing compatibility* trivial. Passing an actual algebra A according to (19) amounts to restricting the class $Mod(SP_r(SP_p))$ to these elements PA for which $par(PA) \simeq A$, i.e., selecting only a "subset" of the semantics of the parameterised specification. If (20) does not hold, this subset may happen to be empty.

Remark. Alternatively to definition 6.1, we might view the semantics of a parameterised specification as $[\![SP_r(SP_p)]\!] \subseteq Mod(SP_p) \times Mod(SP_r(SP_p))$. In the extreme case, we would take the whole class. Persistence would require for all $\langle A, PA\rangle \in [\![SP_r(SP_p)]\!] : PA|_{\Sigma_p} \simeq A$, which is the usual condition of actual parameter protection

$$[\![SP_r(SP_p)]\!] = \{\langle A, PA\rangle : A \in Mod(SP_p), PA \in Mod(P(SP_r)), PA|_{\Sigma_p} \simeq A\}$$

Passing an algebra A as an actual parameter yields the subclass with pairs having the first component equal to A:

$$[\![SP_r(SP_p)]\!](A) = \{\langle A, PA\rangle : PA \in Mod(SP_r(SP_p)), PA|_{\Sigma_p} \simeq A\}$$

As in our definition, this last set may be empty if SP_r is not persistent. This also makes passing compatibility trivial since the semantics of actual parameter passing is a "subset" of the semantics of parameterised specification.

However, here the semantics of parameterised and non-parameterised specifications differ. As we cannot see any direct advantages of this procedure, we accept definition 6.1.

6.1 A Note on Verification

The two problems to be addressed here concern persistence and verification of the first point in definition 6.2.

As in the case of deterministic specifications, persistence is not decidable. An obvious condition would be that $SP_r(SP_p)$ does not imply any new formulas affecting the models of the parameter, i.e., SP_r is a conservative extension of SP_p:

$$\forall \Phi \text{ } over \text{ } Sig(SP_p) : SP_r(SP_p) \vdash \Phi \Rightarrow SP_p \vdash \Phi \tag{21}$$

If a parameterised specification satisfies this condition we say that it is *conservative* and write $SP_p \lhd SP_r$. By the result from standard logic:

$$SP_p \lhd SP_r \;\Leftrightarrow\; \forall M \in Mod(SP_p)\ \exists N \in Mod(SP_r(SP_p)) : M \sim N|_{\Sigma_p}, \tag{22}$$

where $\sim$ is elementary equivalence.[6] An equivalent condition says that $\forall M \in Mod(SP_p)\ \exists N \in Mod(SP_r(SP_p)) : M \preceq N$, where $\preceq$ denotes elementary extension.[7] This does not guarantee that (20) holds, since M may be only a submodel of some $N|_{\Sigma_p}$. But this means that, in the worst case, there will appear new elements of (some) sorts from Σ_p and that the parameter before passing is logically indistinguishable from the one after passing.

Remark. We know (22) as the result from first order logic and may doubt whether this is applicable to the specifications with predicates on functions which are second order. For this case, we can apply Mostowski collapsing theorem to obtain first order structures where each predicate $P_f(h) : (S \to T) \to Bool$ defines a (subset of a) "functional sort" $(S \to T)$ equipped with the additional function $apply_f : (S \to T) \times S \to T$. If Φ is the formula defining P_f, $apply_f$ is defined by the formula $\Phi^{h(\overline{s})}_{apply_f(h,\overline{s})}$.

We have that conservativity with respect to the formal parameter ensures conservativity with respect to all actual parameters:

[6] $M \sim N$ iff for all closed formulas $\phi : M \models \phi \Leftrightarrow N \models \phi$.

[7] $M \preceq N$ iff $|M| \subseteq |N|$ and for all formulas $\phi(\overline{x})$ and $\overline{a} \in |M| : M \models \phi(\overline{a}) \Leftrightarrow N \models \phi(\overline{a})$.

Lemma 6.8 *If $SP_p \lhd SP_r$ and $SP_p \overset{\kappa}{\rightsquigarrow} SP$, then $\kappa(SP) \lhd SP_r$.*

Proof. Consider first the case $\kappa = id$. Since $Mod(SP) \subseteq Mod(SP_p)$, using (22) we obtain:

$$\forall M \in Mod(SP)\ \exists N \in Mod(SP_r(SP_p)) : M \sim N|_{\Sigma_p},$$

We have to show, that such an N is in $Mod(SP_r(SP))$ and $M \sim N|_\Sigma$. The latter conjunct is trivial since $SP_p \rightsquigarrow SP$ implies $\Sigma = \Sigma_p$. But then, we have that $N \models SP_r(SP_p)$ and $N|_\Sigma \models SP$. Thus, since $Mod(SP_r(SP_p) \cup SP) = Mod(SP_r \cup SP_p \cup SP) = Mod(SP_r \cup SP) = Mod(SP_r(SP))$, $N \models SP_r(SP_p) \cup SP$ implies $N \in Mod(SP_r(SP))$.

The general case of κ follows now trivially, since $SP_p \overset{\kappa}{\rightsquigarrow} SP$ means $SP_p \rightsquigarrow \kappa(SP)$. $\square$

Concerning the verification of the implementation of parameterised specifications, definition 6.2 requires checking the relation for all actual parameters. However, lemma 6.3 tells us that this is not necessary. It means that in order to verify a parameterised implementation $P \circ\!\!\rightsquigarrow P'$, we only have to verify the (non-parameterised) implementation of $P \rightsquigarrow P'$. Since P and P' have the same formal parameter, this essentially means that we only have to verify that body of P' implements the body of P.

6.2 A Parameterised Specification of DFS

To illustrate the use of our formalism we give an example of the DFS specification considered as a parameterised specification over the parameters corresponding, respectively, to the sort of vertices, the pair representation of edges, and the set representation of a graph. Implementing sets by sequences (as it has been done in the preceding sections) we will obtain an implementation of DFS which works on graphs represented by sequences of edges.

Assuming the earlier specifications $SET(E), SEQ(E)$, and a specification $PAIR(E)$ with constructor $\langle \_, \_ \rangle : E \times E \to Pair(E)$, the full version of the DFS-specification is parameterised as follows:

$$
\begin{array}{l}
\text{DFS } \textbf{is } \lambda X : \langle \{E\}, \emptyset, \emptyset \rangle, Y : SET(X), Z : PAIR(X), W : SET(Z) \\
\quad \textbf{enrich } X, Y, Z, W \textbf{ by} \\
\quad\quad \begin{array}{rlrcl}
\mathit{let}: & \multicolumn{4}{l}{Ed = Pair(E), Gr = Set(Pair(E))} \\
\mathcal{F}: & \sqsubset : & E \times Gr & \to & Bool \\
 & ch : & Gr \times E & \to & Set(E) \\
\mathcal{N}: & dfs : & Gr \times E & \to & Gr \\
 & tr : & Gr \times E \times Set(E) \times Gr & \to & Gr \\
\mathcal{A}: & 1. & v \sqsubset \emptyset & \doteq & \mathbf{ff} \\
 & 2. & v \sqsubset S \circ \langle x, y \rangle & \doteq & (v \doteq x \vee v \doteq y \vee v \sqsubset S) \\
 & 3. & ch(\emptyset, v) & \doteq & \emptyset \\
 & 4. & ch(G \circ \langle v, x \rangle, v) & \doteq & ch(G, v) \circ x \\
 & 5. & v \not\doteq y & \Rightarrow & ch(G \circ \langle y, x \rangle, v) \doteq ch(G, v) \\
 & 6. & \mathbf{N}_{dfs}(G, v) & \doteq & N_{tr}(G, v, ch(G, v), \emptyset) \\
 & 7. & \mathbf{N}_{tr}(G, v, \emptyset, T) & \doteq & T \\
 & 8. & S \not\doteq \emptyset \wedge x \doteq N_\sqcup(S) \wedge x \sqsubset T & \Rightarrow & \mathbf{N}_{tr}(G, v, S, T) \doteq N_{tr}(G, v, S \setminus x, T) \\
 & 9. & S \not\doteq \emptyset \wedge x \doteq N_\sqcup(S) \wedge x \not\sqsubset T & \Rightarrow & \mathbf{N}_{tr}(G, v, S, T) \doteq \\
 & & & & N_{tr}(G, v, S \setminus x, N_{tr}(G, x, ch(x), T \circ \langle v, x \rangle))
\end{array}
\end{array}
$$

The *let*... below **enrich** introduces convenient abbreviations for sorts coming from the parameters.

Having established that $SET(E) \overset{\sigma,\equiv}{\rightsquigarrow} SEQ(E)$, lemma 6.4 allows us to conclude $DFS \underset{\sigma,\equiv}{\rightsquigarrow} DFS'$, where DFS' is $DFS(E, SEQ(E), PAIR(E), SET(PAIR(E)))$. Since $N_\sqcup \in \mathcal{N}$ in $SET(E)$ is implemented by the function $fst \in \mathcal{F}$ in $SEQ(E)$, we may make the obvious simplifications (which depend only on the signature):

$$
\begin{array}{l}
\text{DFS' } \mathbf{is}\ \ \lambda X : \langle\{E\},\emptyset,\emptyset\rangle, Y : SEQ(X), Z : PAIR(X), W : SET(Z) \\
\quad \mathbf{enrich}\ \ X, Y, Z, W\ \mathbf{by} \\
\qquad \begin{array}{rl}
let: & Ed = Pair(E), Gr = Set(Pair(E)) \\
\mathcal{F}: & \begin{array}{rrcl}
\sqsubset: & E \times Gr & \to & Bool \\
ch: & Gr \times E & \to & Seq(E)
\end{array} \\
\mathcal{N}: & \begin{array}{rrcl}
dfs: & Gr \times E & \to & Gr \\
tr: & Gr \times E \times Seq(E) \times Gr & \to & Gr
\end{array} \\
\mathcal{A}: & \begin{array}{lrcl}
1. & v \sqsubset \emptyset & \doteq & \mathbf{ff} \\
2. & v \sqsubset S \circ \langle x, y\rangle & \doteq & (v \doteq x \vee v \doteq y \vee v \sqsubset S) \\
3. & ch(\emptyset, v) & \equiv & \epsilon \\
4. & ch(G \circ \langle v, x\rangle, v) & \equiv & ch(G, v)\hat{\ }x \\
5. & v \not\doteq y & \Rightarrow & ch(G \circ \langle y, x\rangle, v) \equiv ch(G, v) \\
6. & \mathbf{N}_{dfs}(G, v) & \doteq & N_{tr}(G, v, ch(G, v), \emptyset) \\
7. & \mathbf{N}_{tr}(G, v, \epsilon, T) & \doteq & T \\
8. & S \not\doteq \epsilon \wedge fst(S) \sqsubset T & \Rightarrow & \mathbf{N}_{tr}(G, v, S, T) \doteq N_{tr}(G, v, S \setminus fst(S), T) \\
9. & S \not\doteq \epsilon \wedge fst(S) \not\sqsubset T & \Rightarrow & \mathbf{N}_{tr}(G, v, S, T) \doteq \\
 & \multicolumn{3}{r}{N_{tr}(G, v, S \setminus fst(S), N_{tr}(G, fst(S), ch(fst(S)), T \circ \langle v, fst(S)\rangle))}
\end{array}
\end{array}
\end{array}
$$

Observe one subtle point concerning this implementation. Implementing $SET(E) \overset{\sigma,\equiv}{\rightsquigarrow} SEQ(E)$ means that ch' in DFS' will return elements of sort $Seq(E)$. Since ch in DFS is a function, ch' must be a function too: for a given graph G (of sort $Set(Pair(E))$) and vertex v, $ch'(G, v)$ must return a unique sequence of v's children. However, this issue is easier than the implementation of $\sqcup$ because ch returns a sequence, so that it suffices that results on equivalent arguments are equivalent (and not necessarily equal): for two equal graphs (sets!) we must only have $G \doteq H \Rightarrow ch'(G, v) \equiv ch'(H, v)$.

In order to give a more constructive definition of ch' we implement (refine) the sort Gr by sequences. From $E \overset{\overline{\sigma}}{\rightsquigarrow} PAIR(E)$ (with the obvious signature morphism $\overline{\sigma}$), and $SET(E) \overset{\sigma,\equiv}{\rightsquigarrow} SEQ(E)$, theorem 6.6 yields $SET(PAIR(E)) \underset{\overline{\sigma}}{\overset{\sigma,\equiv}{\rightsquigarrow}} SEQ(PAIR(E))$, which gives $DFS' \underset{\overline{\sigma};\sigma,\equiv}{\rightsquigarrow} DFS''$:

$$
\begin{array}{l}
\text{DFS" } \mathbf{is}\ \ \lambda X : \langle\{E\},\emptyset,\emptyset\rangle, Y : SEQ(X), Z : PAIR(X), W : SEQ(Z) \\
\quad \mathbf{enrich}\ \ X, Y, Z, W\ \mathbf{by} \\
\qquad \begin{array}{rl}
let: & Ed = Pair(E), Gr = Seq(Pair(E)) \\
\mathcal{F}: & \begin{array}{rrcl}
\sqsubset: & E \times Gr & \to & Bool \\
ch: & Gr \times E & \to & Seq(E)
\end{array} \\
\mathcal{N}: & \begin{array}{rrcl}
dfs: & Gr \times E & \to & Gr \\
tr: & Gr \times E \times Seq(E) \times Gr & \to & Gr
\end{array} \\
\mathcal{A}: & \begin{array}{lrcl}
1. & v \sqsubset \epsilon & \doteq & \mathbf{ff} \\
2. & v \sqsubset S\hat{\ }\langle x, y\rangle & \doteq & (v \doteq x \vee v \doteq y \vee v \sqsubset S) \\
3. & ch(\epsilon, v) & \equiv & \epsilon \\
4. & ch(G\hat{\ }\langle v, x\rangle, v) & \equiv & ch(G, v)\hat{\ }x \\
5. & v \not\doteq y & \Rightarrow & ch(G\hat{\ }\langle y, x\rangle, v) \equiv ch(G, v) \\
6. & \mathbf{N}_{dfs}(G, v) & \equiv & N_{tr}(G, v, ch(G, v), \epsilon) \\
7. & \mathbf{N}_{tr}(G, v, \epsilon, T) & \equiv & T \\
8. & S \not\doteq \epsilon \wedge fst(S) \sqsubset T & \Rightarrow & \mathbf{N}_{tr}(G, v, S, T) \equiv N_{tr}(G, v, S \setminus fst(S), T) \\
9. & S \not\doteq \epsilon \wedge fst(S) \not\sqsubset T & \Rightarrow & \mathbf{N}_{tr}(G, v, S, T)\ \equiv \\
 & \multicolumn{3}{r}{N_{tr}(G, v, S \setminus fst(S), N_{tr}(G, fst(S), ch(fst(S)), T\hat{\ }\langle v, fst(S)\rangle))}
\end{array}
\end{array}
\end{array}
$$

Apparently, we have obtained a deterministic specification, so we might attempt to replace all N_f by corresponding functions. But to do that we would have to prove that DFS'' is deterministic. Instead, we only show that the above specification is implemented by the corresponding deterministic specification DFS''' obtained by replacing all $\equiv$ by $\doteq$ and N_f by f. This step is verified using the relational language – according to (18), we have to show $DFS''' \vdash \nu(DFS'')$. The ν translation (7) of the last four DFS'' axioms looks as follows (axioms 1.-5. remain unchanged):

$$
\begin{array}{llll}
\nu(6.) & \forall r\ \exists r_1 : R_{dfs}(G,v,r) & \Rightarrow & R_{tr}(G,v,ch(G,v),\epsilon,r_1) \wedge r \equiv r_1 \\
\nu(7.) & \forall r : R_{tr}(G,v,\epsilon,T,r) & \Rightarrow & r \equiv T \\
\nu(8.) & S \not\equiv \epsilon \wedge fst(S) \sqsubset T & \Rightarrow & \forall r\ \exists r_1 : R_{tr}(G,v,S,T,r) \Rightarrow R_{tr}(G,v,S\setminus fst(S),T,r_1) \wedge r \equiv r_1 \\
\nu(9.) & S \not\equiv \epsilon \wedge fst(S) \not\sqsubset T & \Rightarrow & \forall r\ \exists r_1,r_2 : R_{tr}(G,v,S,T,r)\ \Rightarrow \\
& \multicolumn{3}{r}{R_{tr}(G,v,S\setminus fst(S),r_1,r_2) \wedge R_{tr}(G,fst(S),ch(fst(S)),T\hat{\ }\langle v,fst(S)\rangle,r_1) \wedge r_2 \equiv r}
\end{array}
$$

The axioms of the DFS''' specification are:

$$
\begin{array}{lrcl}
1d. & v \sqsubset \epsilon & \doteq & \mathbf{ff} \\
2d. & v \sqsubset S\hat{\ }\langle x,y\rangle & \doteq & (v \doteq x \vee v \doteq y \vee v \sqsubset S) \\
3d. & ch(\epsilon,v) & \doteq & \epsilon \\
4d. & ch(G\hat{\ }\langle v,x\rangle,v) & \doteq & ch(G,v)\hat{\ }x \\
5d. & v \not\doteq y & \Rightarrow & ch(G\hat{\ }\langle y,x\rangle,v) \doteq ch(G,v) \\
6d. & dfs(G,v) & \doteq & tr(G,v,ch(G,v),\epsilon) \\
7d. & tr(G,v,\epsilon,T) & \doteq & T \\
8d. & S \not\doteq \epsilon \wedge fst(S) \sqsubset T & \Rightarrow & tr(G,v,S,T) \doteq tr(G,v,S\setminus fst(S),T) \\
9d. & S \not\doteq \epsilon \wedge fst(S) \not\sqsubset T & \Rightarrow & tr(G,v,S,T) \doteq \\
& \multicolumn{3}{r}{tr(G,v,S\setminus fst(S),tr(G,fst(S),ch(fst(S)),T\hat{\ }\langle v,fst(S)\rangle))}
\end{array}
$$

The axiom $E0 : Q \doteq S \Rightarrow Q \equiv S$ makes the verification of the axioms $1.-5, \nu(6.), \nu(7.)$ trivial. Notice that this fact is not so much "borrowed" from $SEQ(E)$ as it is a general requirement on any implementation of $\doteq$. Consequently, its use does not violate the modularity of proofs. Now, the antecedents of axioms $\nu(8.), \nu(9.)$ imply the antecedents of the respective axioms $8d., 9d.$ The succedent of $8d.$ implies $\forall r : tr(G,v,S,T) \doteq r \Rightarrow tr(G,v,S\setminus fst(S),T) \doteq r$, which obviously implies the succedent of $\nu(8.)$. Similarly, the succedent of $9d.$ implies

$$\forall r\ \exists r_1 : tr(G,v,S,T) \doteq r \Rightarrow tr(G,fst(S),ch(fst(S)),T\hat{\ }\langle v,fst(S)\rangle) \doteq r_1 \wedge tr(G,v,S\setminus fst(S),r_1) \doteq r,$$

which trivially implies the succedent of $\nu(9.)$.

By lemma 6.3 we may conclude that $DFS'' \rightsquigarrow DFS'''$ (as parameterised specifications). Using the two previous implementations and lemma 5.1, we get $DFS \underset{\sigma,\equiv;\overline{\sigma};\sigma,\equiv}{\rightsquigarrow} DFS'''$.

7 Conclusion

The paper illustrates the need for distinguishing between underspecification and nondeterminism by two examples. Considering the choice operation in the specification of sets as underspecified excludes some desired implementations. Considering an operation (*dfs*) defined in terms of an iteration over unordered structure (set) as deterministic, makes a natural specification (of the DFS algorithm) yield undesired, in fact incorrect, results.

We have adapted the well known notions suggested for implementation of deterministic data types to the context of nondeterminism, in particular, the basic notion of implementation as model class inclusion. Since nondeterminism is specified by means of the axioms, this implies the possibility of restricting nondeterminism of some operations as a special case of the implementation relation.

We have also extended the notion of hierarchical (parameterised) specifications to the nondeterministic data types, and shown the adequacy of the method on the example where a nondeterministic DFS specification parameterised by a specification of sets was implemented by a deterministic specification parameterised by sequences.

The main contribution lies in the introduction of a method for implementation of nondeterministic data types. This method has been equipped with the verification strategy based on the translation of the high level specifications with nondeterminism into standard language, where one can apply existing tools supporting (semi)automatic theorem proving. In this way the verification burden becomes essentially the same as for the deterministic specifications, modulo the necessity of translating the abstract axioms. This, however, can be easily mechanized.

We have shown how various existing formalisms for nondeterminism can be incorporated into our framework. For the semantics based on relations, multialgebras and sets of functions, we have defined appropriate quotient constructions and demonstrated correctness of the introduced verification criteria for data refinement. We have also observed that the methods relying on the identification of nondeterminism with underspecification, in particular the oracle semantics, pose particular difficulties when applied in the context of data refinement.

The introduced technique is *non-intrusive* in the sense that it affects only the nondeterministic parts of the specification. If the specification happens to describe only deterministic operations, both the specification and the verification technique reduce to the standard (equational) case.

Acknowledgments

Oscar Slotosch deserves our thanks for reading, commenting and preparing the manuscript of this paper.

The first author gratefully acknowledges the financial support received from the Norwegian Research Council (NFR).

References

[BGW80] M. Broy, R. Gnatz, M. Wirsing. *Semantics of Nondeterministic and Noncontinuous Constructs.* LNCS vol. 69, (1980).

[Broy93] M. Broy. Functional Specification of Time Sensitive Communicating Systems. ACM Transactions on Software Engineering and Methodology 2:1, Januar 1993, 1-46.

[BW81] M. Broy, M. Wirsing. On the Algebraic Specification of Nondeterministic Programming Languages. *CAAP'81*, LNCS vol. 112, (1981).

[GB80] J. Goguen, R. Burstall. *CAT: a system for the structured elaboration of correct programs from structured specifications.* Tech. Rep. CSL-118, SRI International, (1980).

[Hes88] W.H. Hesselink. A Mathematical Approach to Nondeterminism in Data Types. *ACM Transactions on Programming Languages and Systems*, 10, pp.87-117, (1988).

[Hoo92] P. Hoogendik. Relational programming laws in Boom hierarchy of types. *Proc. of Mathematics of Program Construction*, Springer, (1992).

[Hus93] H. Hussmann. *Nondeterminism in Algebraic Specifications and Algebraic Programs.* Birkhäuser Boston, (1993).

[KW94] V. Kriaučiukas, M. Walicki. Reasoning and Rewriting with Set-Relations I: Ground Completeness, *CSL'94*, (1994). [also Tech.Rep. no.94, University of Bergen, Dept. of Informatics, (1994)]

[Mes92] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency, *TCS*, vol. 96, (1992). [also Tech. Rep. SRI-CSL-90-02, SRI International (1990)]

[Mos89] P.D. Mosses. Unified Algebras and Institutions, *Proc. of LICS'89*, (1989). [also Tech.Rep. DAIMI PB-274, Dept. of CS, Aarhus University, (1989)]

[QG93] X. Qian, A. Goldberg. Referential Opacity in Nondeterministic Data Refinement. *ACM LoPLaS*, vol. 2, no. 1-4: 233-241, (1993).

[ST88] D. Sannella, A. Tarlecki. Toward Formal Development of Programs from Algebraic Specifications: Implementations Revisited. *Acta Informatica*, 25, pp.233-281, (1988).

[Sta89] E. Stark. Compositional relational semantics for indeterminate dataflow networks. *Category Theory and Computer Science*, LNCS vol. 389, (1989).

[Sub81] P. Subrahmanyam. Nondeterminism in Abstract Data Types. *LNCS* vol. 115, (1981).

[Voe91] E. Voermans. Pers as types, inductive types and types with laws. *Proc. of PHOENIX Seminar and Workshop on Declarative Programming*, Springer, (1991).

[Wal93] M. Walicki. *Algebraic Specifications of Nondeterminism.* Ph.D. thesis, Institute of Informatics, University of Bergen, (1993).

[WM95a] M. Walicki, S. Meldal. A Complete Calculus for Multialgebraic and Functional Semantics of Nondeterminism. To appear in *ACM ToPLaS*, (1995).

[WM95b] M. Walicki, S. Meldal. Multialgebras, Power Algebras and Complete Calculi of Identities and Inclusions. 10-th ADT-W, *Recent Trends in Data Type Specification*, LNCS vol. 906, (1995).

[WM95c] M. Walicki, S. Meldal. Generated Models and ω-rule: the nondeterministic case. To appear in Proc. of *TAPSOFT'95*, (1995).