Initiality + Nondeterminism ⇒ Junk

Michal Walicki Sigurd Meldal

1. Introduction

There is no need to argue for the plausibility of the initial semantics. It has immediate intuitive appeal in that it establishes a very close relation between the syntax of a specification and its semantics. Initial semantics is essentially *syntactic*, as reflected in the names "term model" or "word model". Initiality implies that the semantic changes induced by syntactic changes in a specification are minimal, and so reflect the latter in the natural and expected manner. For instance, adding a new sort and a constant of this sort has no exotic influence on the model. The model will be extended with a new sort and just the one element needed for interpretation of the new constant. Also, writing different terms one usually intends them to be distinct entities unless they are explicitly equated with other terms.

The two slogans expressing these properties of initial models are "*no junk*" and "*no confusion*" – no element appears in the model whose existence is not required by the interpretation of terms introduced in the specification, and distinct terms are interpreted as distinct elements unless their identity follows from the equations of the specification.

A limitation of the initial approach is that not all theories have initial models. It is well known [10,8,14] that Horn theories are the most general theories always possessing initial models. As the interest in initial semantics was carried over to the domain of nondeterministic specifications, so was the interest in Horn theories .

This paper is intended as an empirical investigation of the consequences of applying the theory of initiality to the new domain of *nondeterminism*. The presentation is rather informal. We will present a simple, yet illustrative, example and work it out in different formalisms. The models we will consider are *multialgebras* [3,4,7], *computations* [15], and *unified algebras* [11,12]. In each case we will give a short description of the language used. We will have in mind only initial models, and the interpretation of a ground term t in such a model will be denoted $\mathcal{I}(t)$ or $\underline{t}$.

The example itself was used in [11], but has certainly been contemplated by anybody devoting a few minutes' thought to nondeterminism. We are particularly thankful to Vladimir Antimirov for his comments and discussion of it in the context of unified algebras.

Having explored the example using multialgebras, computations and unified algebras, and compared the result of underspecifying a term with the consequences of having only nondeterministic operations, we comment briefly on the relation between the three formalisms. We will conclude that, since nondeterminism subsumes underspecification, the initial models of nondeterminism introduce intuitive junk and, furthermore, that treating nondeterministic specification not merely as underspecification but as specification of (truly) nondeterministic operations does not make the situation worse. Then we suggest that the way to avoid the implausibilities is simply to allow the specifier to say what he intends to say.

2. Underspecification vs. nondeterminism.

2.1. Underspecification

One of the aims of specifying a program is to express its functionality at an abstract level, avoiding complications implied by operational and implementation details. Abstract properties of the intended structure may allow different implementations, and one should avoid overspecifying the problems as this may exclude some, otherwise acceptable, implementations. Limiting a language to Horn formulae may restrict the possibility of assuming such a "don't care" attitude. The paradigmatic example of this makes use of disjunction: assume that we want an operation $c{:}S{\times}S{\rightarrow}S$ which, for arguments a,b, should equal either a or b. The only way to capture this intention without using disjunction is to leave c partly specified or unspecified. For instance:

SP **is**

S: { S }

$$
\begin{array}{llll}
\mathbf{F}: \{ & a,b,h: & & \to S \\
& c: & S\times S & \to S \\
& P: & S & \to Bool \ \}
\end{array}
$$

$$
\begin{array}{lll}
\Pi: \{ & 1.\ P(a) = P(b) = T & 4.\ c(x,x) = x \\
& 2.\ P(h) = F & 5.\ c(x,y) = c(y,x) \\
& 3.\ x=c(a,b) \Rightarrow P(x)=T & 6.\ c(\ x,c(y,z)\ ) = c(\ c(x,y),z\ ) \quad \}
\end{array}
$$

end

where a, b, and h are the three intended elements of sort S. Axiom 3. excludes h from being the result of $c(a,b)$ (provided that T≠F).[1] 4.-6. are included primarily to simplify the model below. They assert general properties of c as join. Underspecification means that the specification possesses several nonisomorphic models, and all of them are taken as the semantics of the specification. SP will allow models where $c(a,b) = a$, and others where $c(a,b) = b$. Unfortunately, loose semantics will typically admit structures which violate the demands of "no junk" and "no confusion". Not only will SP have models where $a=b$, but also such where the result of $c(a,b)$ is neither a nor b.

Restricting the loose semantics to generated models and, in particular, to the initial model of SP presents us with another dilemma. The initial model will contain

x	$\mathcal{I}$(x)	$\mathcal{I}$(P(x))
a	$\underline{a}$	$\underline{T}$
b	$\underline{b}$	$\underline{T}$
h	$\underline{h}$	$\underline{F}$
c(a,b)	$\underline{c(a,b)}$	$\underline{T}$
c(a,h)	$\underline{c(a,h)}$	$\underline{P(c(a,h))}$
c(b,h)	$\underline{c(b,h)}$	$\underline{P(c(b,h))}$

The problematic point is the presence of the element $\underline{c(a,b)}$. Formally, this is quite correct since the corresponding term exists in the specification and is not provably equal to a or b. However, with respect to the original intention it represents *junk*, since a, b, and h were supposed to generate all the elements of the sort S. We may call this "*intuitive junk*" to distinguish it from unreachable elements corresponding to (formal) junk. It may appear in a carelessly written specification, but here it appears because of the insufficiently expressive language.

Thus, due to the limitations of language, underspecification gives us a choice between an imprecisely defined class of models or intuitive junk.

2.2. Nondeterminism

The success of initial semantics for deterministic specifications motivates the search for such semantics also for specifications involving nondeterminism. This implies, in particular, that one should continue using conditional equations (or, if necessary, Horn clauses) involving nondeterministic terms and try to construct initial models for such theories. Typically, nondeterministic operations are modeled by some form of collections (set union, join) of their possible results. This requires explicit distinction between deterministic terms, which are the basic elements of a model, and the nondeterministic ones, which are constructed from the individual values.

Having such a distinction in the language, specifications admitting nondeterministic implementations subsume underspecification as a special case. For instance, let c be a deterministic constant, and $a \sqcap b$ be a nondeterministic choice between a and b. Relating the two terms (by means of equality, inclusion or membership – depending on which primitive predicates are part of the specification language) will amount to saying that c is to be equal a or b, i.e., to underspecifying c.

A conservative approach would use nondeterminism merely for the purpose of underspecification and hope that initiality in classes of models of such specifications would retain the purity expected of traditional initial models. We will argue that, from the point of view of the initial semantics, the sole problem of nondeterminism is caused by the fact that it subsumes underspecification. Nondeterminism in

---

[1] Its formulation reflects the intended structure of the specification with a, b, and h as actual constants and $c(a,b)$ as a defined operation. Writing $P(c(a,b))$=T instead would not change anything in the following argument.

itself does not cause much harm – it is the fact that deterministic operations can be underspecified by means of nondeterministic ones that leads to some intuitive flaws in the initial models.

3. Multialgebras

Nondeterminism can be introduced into a language by two basic means: either by including some primitive operation (choice) with predefined semantics reflecting the intended nondeterminism, or by introducing some new primitive relation/predicate (set membership, inclusion) allowing specification of nondeterministic operations.

3.1. Choice and set equality

Multialgebras represent the first, and still most common, approach to defining algebraic semantics of nondeterminism (e.g.,[3,4,7,13]). Since nondeterministic operations are capable of returning several different results at different invocations, it seems natural to attempt a generalization of equational specifications by modeling operations as *set-valued functions*. The corresponding generalization of equality will interpret it as equality of the result sets, rather than single values. The most straightforward approach is to define a language with two primitives: (1) set equality, and (2) choice operator interpreted as the set of its arguments [6,9]. Unfortunately, the expectation that such a purely equational language will guarantee existence of an initial model turns out to be futile:

Example 3.1

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

□

The problem arises from the fact that one has in advance decided what the meaning of $\sqcap$ should be. Since it *is* set union, and since equality *is* set equality there is no other way to make $\mathcal{I}(0\sqcap 1)$ and $\mathcal{I}(0\sqcap(1\sqcap 2))$ equal than by identifying 2 with 1 or 0. Taking the quotient of the term structure or interpreting $0\sqcap 1$ as the set $\{\underline{0},\underline{1},\underline{2}\}$ are excluded. Axioms affecting the semantics of choice may introduce inconsistency with this predefined meaning (though not logical inconsistency). We have not encountered any results stating the general conditions for existence of initial models in this approach.

3.2. Inclusion

Obviously, set equality is identical to two way inclusion, so removing equality and introducing inclusion as the primitive results in a more flexible language. In order to distinguish values from *sets* of values one also needs an additional determinacy operator. It can be introduced explicitly (as a Det-predicate in [4]), or as a side effect of a reinterpretation of equality (as in [15]). We will consider the former which is less general but sufficient for our purpose here. Nondeterministic operations can be specified by describing their results, and a primitive nondeterministic operation is not needed. The language with (1) conditional formulae constructed from (2) inclusion and (3) a Det-predicate will be referred to as $\mathcal{L}_{heq}$. Turning *c* into a constant and using nondeterministic choice to underspecify it leads to the following specification:

SPM **is**

S: { S }
F: { a,b,c : $\rightarrow$ S
_⊓_ : S×S $\rightarrow$ S }
Π: { 1. Det(a) Det(b) Det(c)

---

[2] Actually, this depends on the definition of homomorphism we are working with. Some definitions could give possibility of interpreting them as arbitrary sets. They had to be different sets, though, and so the problem would persist anyway.

2. $x \prec x \sqcap y \quad y \prec x \sqcap y$

3. $c \prec a \sqcap b$ }

end

Axioms 1. make all three constants deterministic, and axioms 2. make $\sqcap$ capable of returning any of its arguments. (We are using $\prec$ as the syntactic operation with set inclusion as its semantic interpretation.) Axiom 3. ensures then that *c* will be a possible result of choosing among *a* and *b*. Terms are again interpreted as sets with the restriction that a term *t* satisfying *Det(t)*, must be interpreted deterministically, i.e., as a 1-element set.

As in the previous case, initial multialgebras for specifications in $\mathcal{L}_{heq}$ do not necessarily exist. In [4], Hußmann has identified necessary and sufficient conditions ensuring the existence of initial semantics. The example specification satisfies these and its initial model is given by the word multistructure W_Σ:

$$(W_\Sigma)_S = \{\underline{a}, \underline{b}, \underline{c}\}$$

$$\begin{array}{lclrcl} \mathcal{I}(a) & = & \{\underline{a}\} & \mathcal{I}(a \sqcap b) & = & \{\underline{a},\underline{b},\underline{c}\} \\ \mathcal{I}(b) & = & \{\underline{b}\} & \mathcal{I}(a \sqcap c) & = & \{\underline{a},\underline{c}\} \\ \mathcal{I}(c) & = & \{\underline{c}\} & \mathcal{I}(b \sqcap c) & = & \{\underline{b},\underline{c}\} \\ & & \text{for other } t,s \in W_\Sigma\text{:} & \mathcal{I}(t \sqcap s) & = & \mathcal{I}(s) \cup \mathcal{I}(t) \end{array}$$

Fig. 3.2

Although we would like to identify $\underline{c}$ with either $\underline{a}$ or $\underline{b}$ we cannot do so in the initial model because it would exclude existence of homomorphisms to structures realizing the other possibility. Consequently, *c* must be interpreted as a distinct element. “Junk” obeys the general Murphy’s law: doing nothing merely increases the amount of problems! Having admitted $\underline{c}$, axioms 2. and 3. make it appear in the result sets of other operations as well, in particular, $\underline{a} \sqcap \underline{b}$ now has to return not only $\underline{a}$ and $\underline{b}$ but also $\underline{c}$.

Removing *c* and axioms *Det(c)* and 3. from SPM – i.e., removing the underspecified deterministic operation and retaining only nondeterministic choice – would give the expected (and reasonable) initial model with $\mathcal{I}(S) = \{\underline{a},\underline{b}\}$ and $\mathcal{I}(a \sqcap b) = \{\underline{a},\underline{b}\}$.

4. Computations

In [15] we defined a refinement of multialgebraic semantics. Instead of gathering all possible results of nondeterministic operations into one multimodel, the semantics of a specification is the collection of all possible “computations” – each realizing different possibilities of nondeterministic operators. Here we merely sketch the basic idea of this construction in sufficient detail to allow us to present the initial computation for our example.

The specification language is $\mathcal{L}_{heq}$, i.e., the same as for the multialgebras. A computational interpretation of terms and formulae is obtained by extending the model with a new *index* sort and turning all operations into functions by equiping them with an additional index argument. The Det-predicate corresponds then to the fact that the index argument does not influence the value returned by the function. *Det(t)* is modeled by imposing the equality $t_i = t_j$, for all index values *i* and *j*, and so the determinacy axioms allow us to ignore the indices of deterministic terms. Inclusions are modeled by introducing new Skolem functions for all indices in their right hand sides and imposing the corresponding equality. Thus, the axiom $s \prec t$ will lead to the validity of $s_i = t_{\sigma(i)}$. The following three equations would be valid in the computations of SPM as consequences of the axioms 2.-3.:

2′. $x = x \sqcap_{\sigma(x,y)} y \qquad y = x \sqcap_{\delta(x,y)} y$

3′. $c = a \sqcap_\gamma b$

The multialgebraic interpretation of an $\mathcal{L}_{heq}$ term corresponds to collecting all values returned by the corresponding function when the index arguments vary over the whole index sort. E.g., interpreting a term *f(t)* in a computational structure *C* would give $\mathcal{I}(f(t)) = \{ f^C_{\underline{i}}(\underline{t}) \mid \underline{i} \in I^C, \underline{t} \in \mathcal{I}(t) \}$. The initial computation of SPM will be given by the following word structure:

$$\begin{array}{lclcl} \mathcal{I}(a) = \{\underline{a}\} & \mathcal{I}(a \sqcap b) & = & \{ \underline{a} \sqcap_{\sigma(\underline{a},\underline{b})} \underline{b},\ \underline{a} \sqcap_{\delta(\underline{a},\underline{b})} \underline{b},\ \underline{a} \sqcap_\gamma \underline{b} \} & \cup \{ \underline{a} \sqcap_j \underline{b} \mid j \neq \gamma, j \neq \sigma(\underline{a},\underline{b}), j \neq \delta(\underline{a},\underline{b}) \} \\ \mathcal{I}(b) = \{\underline{b}\} & & = & \{ \underline{a},\ \underline{b},\ \underline{c} \} & \cup \{ \underline{a} \sqcap_j \underline{b} \mid j \neq \gamma, j \neq \sigma(\underline{a},\underline{b}), j \neq \delta(\underline{a},\underline{b}) \} \end{array}$$

$\mathcal{I}(c) = \{\underline{c}\}$ $\mathcal{I}(a\sqcap c) = \{ \underline{a}\sqcap_{\sigma(\underline{a},\underline{c})}\underline{c},\ \underline{a}\sqcap_{\delta(\underline{a},\underline{c})}\underline{c} \} \cup \{ \underline{a}\sqcap_j\underline{c} \mid j\neq\sigma(\underline{a},\underline{c}), j\neq\delta(\underline{a},\underline{c}) \}$

$= \{ \underline{a},\ \underline{c} \} \cup \{ \underline{a}\sqcap_j\underline{c} \mid j\neq\sigma(\underline{a},\underline{c}), j\neq\delta(\underline{a},\underline{c}) \}$

.... etc.

Unlike for multialgebras, where additional restrictions were needed to ensure existence of initial models, no restrictions are needed in the case of computations. Every $\mathcal{L}_{heq}$ specification possesses initial computations. But as we can see, computations introduce even more redundancy than multialgebras do, due to the multiplicity of the index terms. These extra elements can be removed by the introduction of additional axioms. For instance, the axiom $\delta(x,y)=\delta(y,x)$ would eliminate all "junk" of the form $y\sqcap_{\delta(x,y)}x$ for all $x,y\in|C|$. But such axioms cannot be introduced at the level of $\mathcal{L}_{heq}$ and, furthermore, their form will depend heavily on the actual axioms of the specification. There is little hope that they can be generated automatically. Even if they could be used for eliminating some junk, there is no way to get rid of the element $\underline{c}$ in the initial computation. As in the case of multialgebras, it cannot be identified with $\underline{a}$ nor $\underline{b}$ since any choice would exclude the existence of homomorphisms to the computation realizing another possibility.

Unlike for multialgebras, the junk here is not merely the consequence of the underspecified c. Removing it (and axiom 3.) would significantly reduce the amount of garbage but would still leave us with redundant elements in the interpretation of $a\sqcap b$.

5. Unified algebras

Unified algebras have been introduced by Peter Mosses in [11,12]. In the following discussion the definitions and notation for unified algebras have been modified to be consistent with the rest of the paper.

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

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

"Unified" refers to the fact that sorts (just like individuals) are elements of the lattice and not sets of the elements as in the case of multialgebras. The partial order of the lattice corresponds to set inclusion, and nondeterministic choice is interpreted as joins.

The operations from Ω can be specified using Horn clauses alone, and it is shown that specifications with Horn clauses always possess initial unified models [12]. The example specification becomes:

SPU **is**
 F: { a, b, c }
 Π: { 1. a:a b:b
  2. c: a⊓b }
end

(Since unified algebras have only one sort there is no need to specify profiles of the operations, except for the number of arguments they take.) The first axiom makes a and b individuals, i.e., deterministic constants. The second one makes c an individual which, in addition, lies below the join of a and b.

Having c as a constant in the signature, one may expect to find it also in the initial model. However, since any model must be a (distributive) lattice, there are more surprises than that. The initial unified algebra for SPU is given by the following figure:

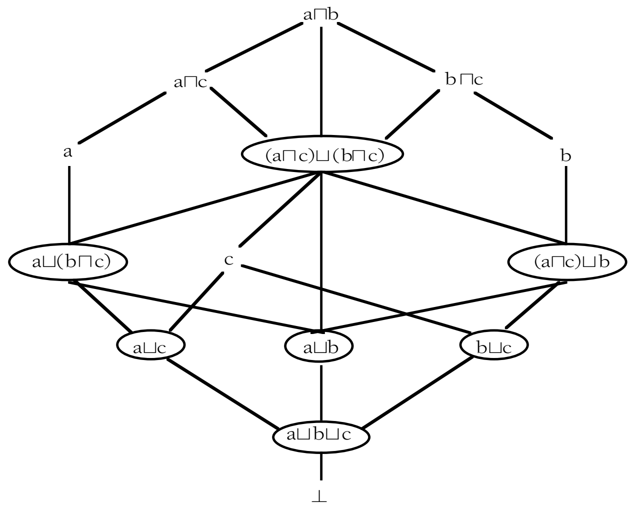

Fig. 5.1

The elements framed with rings certainly do not represent anything we had in mind when writing SPU. They have to appear because the lattice model requires all joins and meets to be present and they are irreducible with the lattice axioms alone.[3] Just as we did in the case of computations, we can get rid of some of the "vacuous sorts" by introducing additional axioms. In the present case, the axiom 3. $a \sqcup b = \perp$ will identify most of the redundant elements leading to the following, more intuitive model:

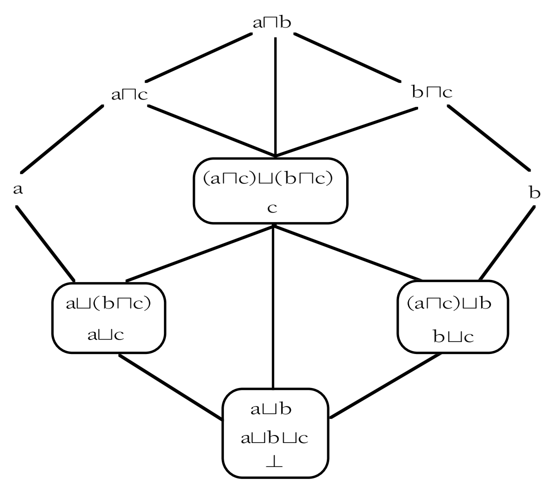

Fig. 5.2

The boxes show which elements from the previous picture have been identified. Here, *c* is still a separate element which cannot be set equal to *a* nor *b*. Consequently, both joins, $a \sqcap c$ and $b \sqcap c$, and meets, $a \sqcup c$ and $b \sqcup c$, must be present as well although we know that one from each pair will actually collapse (e.g., if *c* is equal to *a* then $a \sqcap c = a \sqcup c = a$). We can give some plausibility to the presence of joins – not knowing whether *c* is *a* or *b*, we do not know whether $a \sqcap c$ is going to be $a \sqcap a$ or $a \sqcap b$, i.e., we have to keep both. Thus, in general, even after spending time and effort on finding out which additional axioms are needed to remove irrelevant elements, we will be left with some redundant objects in the initial model. Nevertheless, in unified algebras the user may directly extend the specification with the needed axioms without going via an underlying, indexed specification as was the case with computational semantics.

---

[3] For the purpose of modeling nondeterminism the meets are not needed and the upper semi-lattice would provide sufficient structure. This would give a significantly more intuitive model – see (the upper part of) the next figure. Nevertheless, even for such a simple example, junky *c* and its joins would remain.

Again removal of the axiom $c{:}a\sqcap b$ would lead to the plausible initial model with the elements {a,b,a⊓b,a⊔b,⊥}.

There is another aspect of the "redundancy" in the context of unified algebras. Even if we do not specify explicitly the choice operator, it will be present in the model. In the above example we only said that c should be a result of $a\sqcap b$ and the elements corresponding to $a\sqcap c$ and $b\sqcap c$ have been included in the model, so to speak, automatically. This is, of course, a consequence of the fact that choice is a primitive operator and its semantics is built into the general framework.

6. Comparison

In this section we emphasize some points concerning the relations between the three models we have examined.

It is easy to see that the upper part of the lattice from fig.5.2 is essentially the same as the multimodel W_Σ from fig.3.2 where the partial order is specialized to set inclusion. In particular, the "redundant" elements $\underline{c}$, $\{\underline{a},\underline{c}\}$, $\{\underline{b},\underline{c}\}$ are present in the multimodel as they are in the unified algebra. Certainly, the advantage of W_Σ is that it is obtained directly, while unified algebras required an additional axiom 3. to remove many of unintended elements. On the other hand, unified algebras guarantee the existence of initial models for all specifications with Horn clauses, while only a subclass of Horn specifications possess initial multimodels.

Similar remarks apply to the relation between initial computations and multimodels. All Horn specifications have initial computations, but, as we have seen, such computations introduce a lot of "junk" and, in order to remove it, one has to consider the indexed structure and impose additional axioms on it.

Thus, one always has initial computations and initial unified algebras, but this generality implies problems with "eliminable junk". Conditions which specifications must satisfy in order to allow initial multimodels, on the other hand, seem to lead to simpler and more intuitive semantics. Nevertheless, even the "most natural" model W_Σ involves elements which are superfluous when compared to the intentions of the specifier.

One of the main differences between unified algebras and the other two semantics is that in the former choice is a primitive of the language. This is plausible in so far as choice is a very natural nondeterministic operation which will typically be present in specifications using nondeterminism. Although in the initial multimodel of SPM choice happens to be join, this will not be the case in general. In other (non-generated) multimodels $a\sqcap b$ may return all other kinds of elements besides a and b. The same applies to, for instance, $a\sqcap c$. Thus, all models of SPM will satisfy $a\sqcap c \prec a\sqcap(a\sqcap b)$, but not necessarily $a\sqcap c \prec a\sqcap b$. The latter simply does not hold in the models where $a\sqcap c$ returns some elements not returned by $a\sqcap b$. This, of course, does not happen in the unified algebra models for which nondeterminism cannot be defined by the user.

The example in section 5 indicated that using a predefined semantics of some nondeterministic operator may have its drawbacks. It is precisely this predefined meaning of ⊓ in unified algebras which is responsible for the occurrence of "junk" which, to some extent, can be eliminated by additional axioms. $\mathcal{L}_{heq}$, on the other hand, does not restrict the user in this respect but gives him the possibility of explicitly specifying the join operator if there is a need for it. Inclusion of the axioms 4.-6. from SP (with ⊓ for c) into SPM (i.e., two new axioms with inclusions per one equational axiom from SP) would make ⊓ into join and result in the validity of $a\sqcap c \prec a\sqcap b$. This gives increased flexibility in specifying nondeterministic operators since each such operator can be specified directly – in unified algebras all nondeterminism must be referred back to the primitive choice operator.

Concluding our examples we can just repeat the sequence of implications:

$$\text{nondeterminism + initiality} \;\Rightarrow\; \text{underspecification + initiality} \;\Rightarrow\; \text{junk.}$$

The superfluous objects may or may not have some intuitive plausibility but are not among the objects usually intended by the writer of the specification. This is, of course, an empirical statement based. We do not claim to have proved that all initial models which can possibly be produced in the future are doomed to contain "junk". But it is reasonable to expect that a high degree of sophistication may be needed in order to cope with the problems indicated by the above examples. Even if one succeeds in eliminating junk, the result may lack the simplicity of the original initial semantics of deterministic specifications.

The examples indicate that the amount of redundancy is less in multialgebras than in the other two models. However, the existence of initial multialgebras is guarded by most restrictive conditions. Unified algebras specified with Horn formulae always posses initial models, but the occurrence, and the character, of redundant elements in such models is difficult for a human being to determine from the specification text alone. The same applies to computations. Although we can eliminate some of "junk" by introducing additional axioms, it can be quite a non-trivial task to find the appropriate axioms.

Initiality is certainly a virtue, but an even greater virtue is the adequacy of the model. It seems that, in the case of nondeterminism, the "no junk" property of initial models gets lost. As a matter of fact, our discussion indicates that initiality becomes a powerful ally of intuitive junk in the semantics of nondeterministic specifications.

7. Disjunction & Quasi-Initiality

All the constructions we have described attempt to indicate that c is equal to a or b, as is the intention of the specification, without actually identifying c with any of the two: the "basic junk" in all the examples was the $\underline{c}$ element. One wants to model the fact that "c=a *or* c=b" without spelling it clearly out. This fear originates, of course, in the knowledge that disjunctive equations do not, in general, allow initial models.

The source of evil lies in the fact that nondeterminism allows underspecification and that one insists on using initial models of such specifications. Deterministic underspecifications give rise to similar problems. However, in the deterministic case underspecification is an optional tool for handling exceptional situations rather than the standard procedure. Typically, one tries to apply some strategy, such as generator induction [1,2,5], resulting in specifications with intended initial models. Nondeterminism, on the other hand, implies disjunction and, with it, underspecification.

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

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

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

The first point refers to the fact that in the lattice model join is an operation (choice) with predefined semantics. If there are several constants in the specification SPU, say d besides a, b and c, then unified algebra is forced to interpret the element $a \sqcap d$ as join. But if we are interested in a binary operation which is only some variation of choice, e.g., chooses nondeterministically merely between a and b, and for other arguments behaves as, say first projection, then we have to relate it to the primitive choice operation. $a \sqcap d$ will always be there. Disjunction allows us to specify different kinds of nondeterministic (also partially defined) operations without any reference to some basic form of nondeterminism.

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

SPD **is**

S: { S }

F: { a,b,c : $\rightarrow$ S

_⊓_: S×S $\rightarrow$ S }

Π: { 1. Det(a) Det(b) Det(c)

2. $x \prec x \sqcap y$ $y \prec x \sqcap y$

3. $c=a \vee c=b$ }

end

Notice that the choice operator is actually not needed at all. Disjunction allows us to distinguish *c* as underspecified in SPD from *d* as a result returned by nondeterministic choice in SPD1 below (although we are not going into the details here – see [15]).

SPD does not have an initial model. Nevertheless, disjunction does not spoil initiality completely. We have shown that disjunctive specifications always possess quasi-initial computational semantics, and that mild restrictions on the specifications guarantee existence of quasi-initial multialgebra semantics [15]. Quasi-initiality generalizes initiality to the situations involving "either ... or ...". For SPD, such a semantics can be summarized by saying that the model class will consist of two parts: the models where *c* equals *a* (*Ma*) and those where it equals *b* (*Mb*). Two models, *A* and *B* given by:

$$\begin{array}{ccccc} A & & & & B \\ \underline{a} & \longleftarrow & a & \longrightarrow & \underline{a} \\ \underline{b} & \longleftarrow & b & \longrightarrow & \underline{b} \\ \{\underline{a},\underline{b}\} & \longleftarrow & a \sqcap b & \longrightarrow & \{\underline{a},\underline{b}\} \\ \underline{a} & \longleftarrow & c & \longrightarrow & \underline{b} \end{array}$$

Fig. 7.1

will be initial in the respective components *Ma* and *Mb* of the model class.

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

SPD1 **is**

S: { S }

F: { a,b,c,d : $\rightarrow$ S

_⊓_: S×S $\rightarrow$ S }

Π: { 1. Det(a) Det(b) Det(c) Det(d)

2. $x=y \sqcap z \Rightarrow x=y \vee x=z$

3. $c=a \vee c=b$

4. $d \prec a \sqcap b$ }

end

Axiom 2. specifies binary choice by underspecifying each particular application of it. It reads: if *x* is a result of (some) application of $y \sqcap z$ then *x* equals *y* or *z*. (Compare this to the axiom 2. from SP.) Substituting *a*, *b*, *d* for *y*, *z*, *x*, and using axiom 2. we will get that *d* equals *a* or *b*. The initial covering of the model class of SPD1 will contain the following models. Each of the models *A* and *B* from fig.7.1 will now be split into two new models depending on whether $\mathcal{I}(d)=\underline{a}$ or $\mathcal{I}(d)=\underline{b}$. In addition, there will be models where $\mathcal{I}(a \sqcap b) = \{\underline{a}\}$, and $\mathcal{I}(a \sqcap b) = \{\underline{b}\}$ (these can be excluded by adding axioms 2. from SPD). Since *d* is defined in terms of $a \sqcap b$, in models where one of these hold *d* will be equal to the respective $\mathcal{I}(a \sqcap b)$. *c* remains underspecified and so even if $\mathcal{I}(a \sqcap b)=\{\underline{a}\}$ *c* may still equal *b*. We believe that all these models reflect the intuitive possibilities of the specification. In particular, none will contain any redundant elements, and the sort S will consist of at most $\underline{a}$ and $\underline{b}$.

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

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

[14] Tarlecki, A., "Free constructions in algebraic institutions", in *Mathematical Foundation of Computer Science '84*, vol. 176, LNCS, Springer, 1984.

[15] Walicki, M., ALgebraic Specifications of Nondeterminism, Dr.Scient. Thesis, Universitetet i Bergen, Institutt for Informatikk, 1993.