ON THE ALGEBRAIC SPECIFICATION OF NONDETERMINISTIC PROGRAMMING LANGUAGES *)

M. Broy, M. Wirsing **)
Technische Universität München, Institut für Informatik
Arcisstraße 21, D-8000 München 2

Abstract

Different semantic models for a nondeterministic programming language are defined, analysed, and compared in the formal framework of algebraic specifications of programming languages by abstract types. Four abstract types are given representing *choice ("erratic") nondeterminism*, *backtrack ("demonic") nondeterminism*, *unbounded ("angelic") nondeterminism* and *loose nondeterminism*. The classes of algebras of these types represent classes of semantic models. A comparison of these classes of semantic models shows the connections and differences between the four different concepts of nondeterminism as found in programming languages.

1. Introduction

The concepts of nondeterminism and nondeterminacy have found their way into programming languages only during the middle of the last decade, although McCarthy in his pioneering paper /McCarthy 63/ already introduced an "ambiguity operator" and Floyd in /Floyd 67/ suggested nondeterministic programs for the implicit formulation of backtrack programs.

Recently the growing interest in rigorous methods for formal specification and program development and numerous attempts to define a formal semantics for concurrent programming languages has led to intensive investigations in the theory and formal foundations of nondeterminism. However, a careful study of the different approaches indicates, that not only the formal description methods are different, but there are actually different concepts described, although the differences are often rather

*) This work was carried out within the Sonderforschungsbereich 49 - Programmiertechnik - Munich

**) Present address: Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ

sophisticated but nevertheless of great importance. Strictly speaking essentially ("extensionally") different semantic models can be given for nondeterministic programming languages reflecting the different concepts of nondeterminism.

Recent studies have shown, that algebraic methods allow the specification of programming languages by abstract (data) types in a short, flexible way (cf./Broy, Wirsing 80a/). There the *context free syntax* corresponds to the signature (the term-algebra represents the set of syntactically correct programs), the *context conditions* (sometimes called "static semantics") are expressed by particular definedness predicates (restricting the term algebra), and the *semantics* is specified by a number of (conditional) equations. Then each model of that type can be considered as a particular semantic model of the programming language. Due to the termination problem of partial recursive functions such an algebraic specification generally includes semantic models where optimal or even maximal fixed points are associated with recursive definitions. The minimality property of least fixed points, however, can be conveniently expressed by *weakly terminal models*, the existence of which is guaranteed under certain (syntactic) conditions (cf. /Broy, Wirsing 80b/).

The class of *extensionally equivalent* models of the type containing the weakly terminal models comprises all possible semantic models which specify the semantics of least fixed points (syntactic, operational, algorithmic and mathematical models). In particular the initial model of the type lies in this class which forms a complete lattice of models (in the usual sense, cf. /Wirsing, Broy 80/).

In this formal framework it is also possible to discuss the semantic models of nondeterministic (applicative or procedural) programming languages. The various concepts of nondeterminism such as *backtrack* nondeterminism versus *choice* nondeterminism (cf. /Broy et al. 80/, /Kennaway, Hoare 80/) as well as *loose* versus *tight* nondeterminism (cf. /Park 80/) may be discussed conveniently in the algebraic approach by the particular classes of models of a nondeterministic programming language characterized by the resp. semantic equations.

We show that *backtrack* nondeterminism, *unbounded* nondeterminism and *choice* nondeterminism admit terminal semantics. The weakly terminal models of *backtrack* nondeterminism as well as of *unbounded* nondeterminism are properly weaker than those of *choice* nondeterminism. In the *(partial) initial semantics* of both forms of nondeterminism nondeterministic statements differ only in their evaluation, while the induced equalities between them are the same.

Loose nondeterminism does not allow terminal or initial semantics, but only minimal models which correspond to all possible deterministic and nondeterministic least fixed point semantics which implement nondeterministic statements. The weakly terminal model of *backtrack* nondeterminism is one of these minimal models. By introducing an "implementation" relation $\subseteq_I$ we can structurize these minimal models in such a way that the $\subseteq_I$ - minimal models are exactly the deterministic least fixed point implementations. The weakly terminal models of *choice* nondeterminism are optimal in the following sense: They are the weakest models which are $\subseteq_I$ - greater than all $\subseteq_I$ - minimal models.

Finally we show that the so-called Egli-Milner Ordering is a consequence of the specification using weak homomorphisms and thus is "natural" in the weakly terminal models.

In fact, the goal of this case study is twofold: First, we want to demonstrate how algebraic methods can be used as a powerful, flexible tool for the formulation and analysis of semantic specifications. Second, we give an attempt to clarify, unify, and compare several notions of nondeterminism with rather sophisticated differences as found in the literature.

We demonstrate our approach by means of abstract data types specifying the sort sta of nondeterministic statements. The types define procedural programming languages very similar to Dijkstra's language of guarded commands. We investigate several closely related versions:

- a type AN the weakly terminal model of which corresponds to *unbounded ("angelic")* nondeterminism (this type resembles to the wlp-calculus definition of Dijkstra).

- a type BN the weakly terminal model of which corresponds to *backtrack ("demonic")* nondeterminism (this type resembles to the wp-calculus definition of Dijkstra).

- a type CN the weakly terminal model of which corresponds to *choice ("erratic")* nondeterminism. Every model of CN implements a model of BN in a "natural" way.*)

- a type LN corresponding to *loose* nondeterminism. For this type there does not exist a weakly terminal model. However all models of AN, BN as well as all models of CN are models of LN, too. Each minimal model of LN represents the mathematical semantics of a particular (possibly deterministic) programming language.

---

*)This type resembles to the wp/wlp-calculus definition of Dijkstra.

2. Basic Definitions

Before we define one type we briefly give the most important definitions (for a complete definition see /Broy, Wirsing 80a/). We consider *hierarchical abstract types* with *primitive subtypes* and *finitely generated partial heterogeneous* Σ-*algebras* as models; i.e. partial heterogeneous Σ-algebras without proper subalgebras. Between two Σ-algebras A und B a family φ of total mappings is called (*partial*) Σ -*homomorphism* (cf./Grätzer 68/), if for all operations f

$$\varphi(f^A(x_1,\ldots,x_n)) = \begin{cases} f^B(\varphi(x_1),\ldots,\varphi(x_n)) & \text{if } f^A(x_1,\ldots,x_n) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

and if

$$f^A(x_1,\ldots,x_n) \text{ defined} \Rightarrow f^B(\varphi(x_1),\ldots,\varphi(x_n)) \text{ defined}$$

A model I of T is called *initial* , if for all models A of T there exists a unique homomorphism $\varphi : I \to A$. An initial model I is *minimally defined*, i.e. every term t which is undefined in some model of T is undefined in I, too.

The properties of homomorphisms for total algebras are generalized by the following notion (cf. /Broy, Wirsing 80b/).

A family φ of partial mappings is called *weak* Σ -*homomorphism*, if for all operations f

$$\varphi(f^A(x_1,\ldots,x_n)) = \begin{cases} f^B(\varphi(x_1),\ldots,\varphi(x_n)) & \text{if } f^B(\varphi(x_1)\ldots,\varphi(x_n)) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

If such a weak Σ-homomorphism exists, then B is called *weaker* than A . A mapping which is both a partial Σ-homomorphism and a weak Σ-homomorphism is called a *strong* Σ-*homomorphism*.

In order to describe observable equivalence we need a notion of terminality for partial algebras. Let I be an initial model of T and consider the class $W =_{def} \{A \mid$ there exists a strong Σ-homomorphism $\varphi : I \to A\}$. Then a model Z of T is said to be *weakly terminal* if Z is strongly terminal in W, i.e. for all $A \in W$ there exists a strong Σ-homomorphism $\varphi : A \to Z$. The weakly terminal models as well as all elements of W are minimally defined.

Let us fix a single model P' of the primitive subtype P of T and consider only the models of T which are extensions of P'. Then every two models A and B for which a strong Σ-homomorphism $\varphi : A \to B$ or $\varphi : B \to A$ exists are *extensionally equivalent*, i.e. for every function f with range in P and every

nonprimitive term t we have $f(...,t,...)^A = f(...,t,...)^B$. In particular, W forms a class of extensionally equivalent models. Every Σ-homomorphism between two extensionally equivalent models is a strong Σ-homomorphism. If C is a class of extensionally equivalent models then a strongly initial (terminal) model $A \in C$ is called *(C-)extensionally* initial (terminal). For example, the initial models of T are W-extensionally initial and the weakly terminal models are W-extensionally terminal (cf. figure 1).

The extensional equivalence leads to another definition of terminality. A model R of T is called *reachable*, if for all models A of T there exists an extensionally equivalent model B of T such that there is a weak Σ-homomorphism $\varphi : B \to R$. Every reachable model is minimally defined, T is *reachably terminal* if it is strongly terminal in the class of all reachable models.

If an initial model exists, then every reachably terminal model is weakly terminal (but in general not vice versa).

A model A of a hierarchical abstract type is called *fully abstract*. (cf. /Milner 77/) if for every pair of terms t1, t2 of nonprimitive sort $t1^A = t2^A$ iff for every primitive context K[x]: $K[t1]^A = K[t2]$; a *primitive context* K[x] for terms of sort s is a term K[x] with the only free variable x such that for every term t of sort s, K[t] is a term of primitive sort.

Obviously (cf. /Broy, Wirsing 80b/) a fully abstract model is minimal with respect to strong homomorphisms. Furthermore, if there exists a fully abstract, minimally defined model of a type and a weakly terminal model, then both are isomorphic. Both notions of minimal full abstractness and weak terminality therefore capture the notion of observable equality or functional equivalence. This means that in a fully abstract model two terms are considered to be equal, iff all observable results of applications of this term (the result of this term in all primitive contexts) are equal. Then the two terms are called *visibly equivalent*.

3. The Abstract Type of Choice Nondeterminism

We define an abstract type comprising the following primitive sorts:

dom , the sort of a semantic objects (including the truth values tt and ff and their characteristic operations) with an equality operation ~ ,

var , the sort of identifiers for programming variables,

proc , the sort of identifiers for procedures,

exp , the sort of arithmetic expressions over var together with a total evaluation function eval : exp → dom , which yields error for free identifiers (where error is a defined constant of dom). We denote by e1[e2/v] the

substitution of v in e1 by e2.

bexp, the set of boolean expressions (also with evaluation function eval).

For simplicity we may assume that these sorts are given by abstract types, which are *monomorphic*, i.e. for which up to isomorphic only one model exists. Equivalently we might assume to take always initial (or terminal) models of the primitive subtype (cf. /Broy, Wirsing 80b/).

As the only nonprimitive sort we specify the sort sta of nondeterministic statements with the *constructor functions:*

$$
\begin{array}{ll}
\text{nop, abort} & : \rightarrow \underline{\text{sta}}, \\
\text{assign} & : \underline{\text{var}} \times \underline{\text{exp}} \rightarrow \underline{\text{sta}}, \\
\text{if} & : \underline{\text{bexp}} \times \underline{\text{sta}} \times \underline{\text{sta}} \rightarrow \underline{\text{sta}}, \\
\text{semi, choice} & : \underline{\text{sta}} \times \underline{\text{sta}} \rightarrow \underline{\text{sta}}, \\
\text{letrec} & : \underline{\text{proc}} \times \underline{\text{sta}} \rightarrow \underline{\text{sta}}, \\
\text{call} & : \underline{\text{proc}} \rightarrow \underline{\text{sta}},
\end{array}
$$

As *semantic functions* we use

$$
\begin{array}{l}
\text{loops} : \underline{\text{sta}} \rightarrow \{\text{tt,ff}\} \\
\text{elem} : \underline{\text{sta}} \times \underline{\text{exp}} \times \underline{\text{dom}} \rightarrow \{\text{tt,ff}\}
\end{array}
$$

with the meaning

loops(S) = ff iff the execution of S cannot lead to a nonterminating computation

elem(S,e,x) = tt iff after the execution of S the expression e may be evaluated to x .

First we specify a number of semantic equalities for statements:

(STA)

$$
\begin{array}{l}
\text{semi(abort,S)} = \text{abort} = \text{semi(S,abort)}, \\
\text{semi(nop,S)} = \text{S} = \text{semi(S,nop)}, \\
\text{semi(semi(S1,S2),S3)} = \text{semi(S1, semi(S2,S3))}, \\
\text{choice(S1,choice(S2,S3))} = \text{choice(choice(S1,S2),S3)}, \\
\text{letrec(p,S)} = \text{S[letrec(p,S)/call(p)]}, \\
\text{semi(if(b,S1,S2),S3)} = \text{if(b,semi(S1,S3), semi(S2,S3))}, \\
\text{semi(assign(v,e), if(b,S1,S2))} = \text{if(b[e/v], semi(assign(v,e),S1),} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{semi(assign(v,e),S2))}, \\
\text{semi(choice(S1,S2),S3)} = \text{choice(semi(S1,S3), semi(S2,S3))}, \\
\text{semi(S1,choice(S2,S3))} = \text{choice(semi(S1,S2), semi(S1,S3))}, \\
\text{if(b,choice(S1,S2),S3)} = \text{choice(if(b,S1,S3), if(b,S2,S3))}, \\
\text{if(b,S1,choice(S2,S3))} = \text{choice(if(b,S1,S2), if(b,S1,S3))}, \\
\text{choice(S1,S2)} = \text{choice(S2,S1)},
\end{array}
$$

We consider the following semantic equations involving the evaluation-operations eval, loops and elem (following /Broy, Wirsing 80b/ to specify the definedness of a term t by DEFINED(t)):

loops(nop) = ff, elem(nop,e,x) = (x ~ eval(e)),

DEFINED(abort), DEFINED(letrec(p,S)), DEFINED(if(B,S1,S2)),

eval(b) = tt ⇒ if(b,S1,S2) = S1,

eval(b) = ff ⇒ if(b,S1,S2) = S2,

eval(b) = error ⇒ if(b,S1,S2) = abort,

loops(semi(S,assign(v,e1))) = loops(S),

elem(semi(S,assign(v,e1)),e2,x) = elem(S,e2[e1/v],x),

loops(semi(S,call(p))) = ff, elem(semi(S,call(p)),e,x) = (x ~ error),

DEFINED(semi(S1,S2)), DEFINED(choice(S1,S2)),

For our choice operation we require

$$(loops(S1) = ff \wedge loops(S2) = ff) \Rightarrow loops(choice(S1,S2)) = ff$$

$$elem(S1,e,x) = tt \Rightarrow elem(choice(S1,S2),e,x) = tt$$

Let us call this type CN. Every statement is defined in every model of CN whereas loops and elem may be partial functions. We indicate the undefinedness of the expression loops(S) by loops(S) = undefined (analogously for elem(S,e,x)). The theorems in /Broy, Wirsing 80a,b/ immediately give the following proposition.

Prop:

(1) The type CN is weakly sufficiently complete and every statement is defined

(2) The type CN has a reachably terminal model C with the following properties:

(a) $C \models loops(S) \in \{ ff, undefined \}$

(b) C is a minimally defined model:

- $\exists$ model M : $M \models loops(S) = undefined \Rightarrow C \models loops(S) = undefined$
- $\exists$ model M : $M \models elem(S,e,x) = undefined \Rightarrow C \models elem(S,e,x) = undefined$

(c) C is a fully abstract model i.e.

$C \models S1 = S2$

iff for all $b \in \{tt,ff,undefined\}$, sta S, exp e, dom x:

$\vdash loops(semi(S,S1)) = b \Leftrightarrow \vdash loops(semi(S,S2)) = b$

and

$\vdash elem(semi(S,S1),e,x) = b \Leftrightarrow \vdash elem(semi(S,S2),e,x) = b$

(3) The type CN has an initial model I_C which is minimally defined. The equality in I_C is determined by the equations STA:

$I_C \models S1 = S2$ iff $STA \vdash S1 = S2$

Therefore two statements are identical in the weakly terminal model C if they are visibly equivalent. From the "minimal definedness"-property we see that the weakly terminal models are equivalent to least fixed point semantics. The weak homomorphisms induce exactly the Egli-Milner-ordering (cf. e.g. /Nivat 80/) between semantic models:

Prop.

Let A, B be models of CN. If there exists a weak homomorphism from A to B then for every statement S

$$S^B \subseteq_{\text{Egli-Milner}} S^A$$

i.e. for all identifiers y and dom x :

$$B \models \text{elem}(S,y,x) = tt \Rightarrow A \models \text{elem}(S,y,x) = tt$$

and $\quad B \models \text{loops}(S) = ff \Rightarrow$

$$(A \models \text{loops}(S)=ff \wedge (B \models \text{elem}(S,y,x) = tt \Leftrightarrow A \models \text{elem}(S,y,x) = tt)) ,$$

The initial model I_C is minimally defined and I_C and C are extensionally equivalent , i.e. for all $b \in \{tt, ff, \text{undefined}\}$

$$C \models \text{loops}(S) = b \quad \text{iff} \quad I_C \models \text{loops}(S) = b$$

and $\quad C \models \text{elem}(S,e,x) = b \quad \text{iff} \quad I_C \models \text{elem}(S,e,x) = b$

The equality between two statements in I_C is the strong equality: Two statements are identical in I_C if their equality is provable from the axioms STA .

The class of minimally defined models of CN coincides with the class of reachable models and forms a complete lattice w.r.t. to the usual homomorphisms as ordering relation (cf. /Wirsing, Broy 80/). The initial model I_C is initial in this class whereas the weakly terminal model is terminal. As in /Broy, Wirsing 80b/ one can define a partial order on the classes of extensionally equivalent models by

C1 $\leq$ C2 iff there exist models M1 $\in$ C1 and M2 $\in$ C2 such that

loops^{M1} and elem^{M1} are "less defined" than loops^{M2} and elem^{M2}

where "less defined" reflects the usual ordering on flat domains (cf. e.g./Manna 74/). Then the minimally defined models are a minimum in this ordering. There does not exist a maximum, but every maximal class corresponds to maximal fixed point semantics (cf. figure 1).

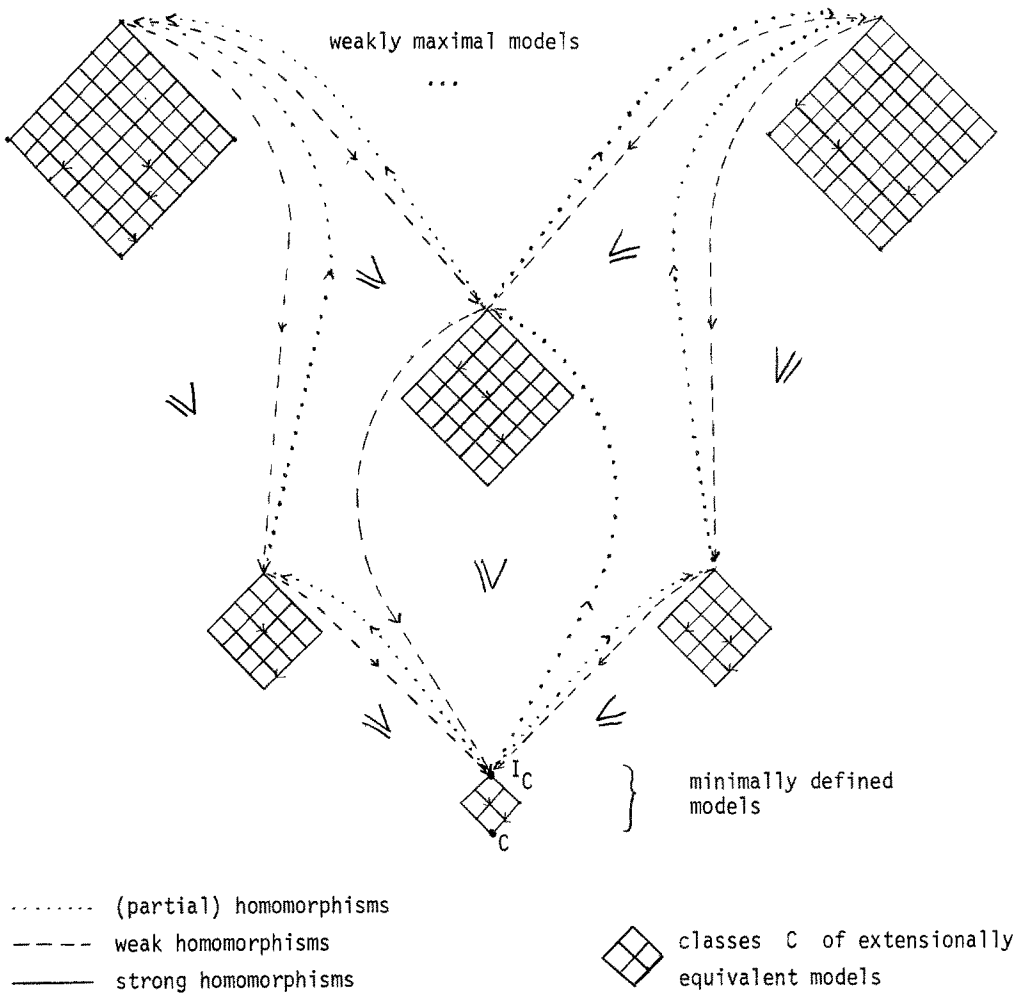

Figure 1: The structure of CN

In particular, for every minimally defined model M we have

$$M \models loops(S) \neq tt,$$
$$M \models loops(S) = ff \Rightarrow elem(S,e,x) \in \{tt,ff\},$$

and

$$M \models loops(S) = undefined \Rightarrow elem(S,e,x) \in \{tt, undefined\}.$$

According to the definition of the Egli-Milner ordering as defined for models we define for nondeterministic statements S1, S2:

$S1 \sqsubseteq_{\text{Egli-Milner}} S2$ iff

$\forall$ sta S : (loops(semi(S,S1)) = ff $\wedge \forall$ exp e, dom x :elem(semi(S,S1),e,x) = elem(semi(S,S2,e,x))
$\vee$ (loops(semi(S,S1)) $\neq$ ff $\wedge \forall$ exp e, dom x :elem(semi(S,S1),e,x) = tt $\Rightarrow$
elem(semi(S,S2),e,x) = tt)

This ordering is used to define a fixed point theory for nondeterministic programs. In minimally defined models of CN the (functionals associated with) recursive procedures are continuous wrt. to the Egli-Milner ordering (cf. /Nivat 80/). In particular this means that if elem(S,e,x) is tt for infinitely many x then loops(S) $\neq$ ff.

4. Backtrack Nondeterminism, Unbounded Nondeterminism and Loose Nondeterminism

Now we specify the further types AN, BN and LN based on the type CN.

```
type BN ≡ sort bsta ,
          bn : sta  → bsta ,
          belem : bsta × exp × dom → {tt,ff} ,
          bloops: bsta →  {tt,ff},
          bloops(bn(S)) = loops(S),
          belem(bn(S),e,x) = (not(loops(S)) and elem(S,e,x)),
          DEFINED(bn(S))
                                                  endoftype
```

```
type AN ≡ sort asta ,
          an : sta → asta,
          aelem : asta × exp × dom → {tt,ff},
          aloops : asta → {tt,ff},
          loops(S) = ff ⇒ aloops(an(S)) = ff,
          aelem(an(S),e,x) = elem(S,e,x),
          loops (S1) = ff ⇒ aloops(an(choice(S1,S2))) = ff,
          DEFINED(an(S))
                                                  end of type
```

Following /Broy, Wirsing 80b/ we use a definedness predicate "DEFINED" to specify the definedness of all nondeterministic statements in the types AN, BN, and LN.

type LN ≡ sort lsta ,
ln : sta → lsta ,
lelem : lsta × exp × dom → {tt,ff} ,
lloops: lsta → {tt,ff},

loops(S) = ff ⇒ lloops(ln(S)) = ff,
lelem(ln(S),e,x) = tt ⇒ elem(S,e,x) = tt ,
(*) lloops(ls) = ff ⇒ ∃ exp e, dom x : lelem(ls,e,x) = tt,
DEFINED(ln(S))

endoftype

Note, that we do not consider the type CN to be part of the types AN, BN and LN but as hidden. The same technique is applied e.g. in /Hennessy, Plotkin 80/. The axiom (*) must be required for LN but it holds in minimally defined models of AN, BN and CN.

The following propositions give some information about the types BN, AN and LN and their relationship to CN :

Prop: (1) The type BN is weakly sufficiently complete and every statement is defined.

(2) The type BN has a reachably terminal model B with the following properties
(a) B is a minimally defined, fully abstract model,
(b) for every two closed statements (i.e. statements without non-initialized variables) S1, S2: $B \models S1 = S2$ iff
$C \models loops(S1) = loops(S2) = undefined$ or $C \models S1 = S2$

(3) For every model N of CN there exists a model M of BN which is weaker than N .

(4) For every model M of BN there exists a model N of CN such that M is weaker than N.

(5) The type BN has an initial model I_B the restriction $I_B|_{STA}$ of which (to the constructor functions of statements) is isomorphic to the restriction $I|_{STA}$ of the initial model of CN

Therefore the equality between statements is the same in the initial models of BN and CN, whereas according to (2) the weakly terminal model B of BN is properly weaker than the weakly terminal model C of CN. The "natural" weak homomorphism $\varphi : N \to M$ (for models N of CN and M of BN) which is defined by

$\varphi(S^N) =_{def} bn(S)^M$, $\varphi(loops^N) =_{def} bloops^M$ and $\varphi(elem^N) =_{def} belem^M$

is a surjective functor from CN onto BN.

Prop.: (1) The type AN is weakly sufficiently complete and every statement is defined.

(2) The type AN has a reachably terminal model A with the following properties

(a) A is minimally defined and fully abstract;

(b) for every two closed statements (i.e. statements without noninitialized variables) S1, S2 :

$A \models S1 = S2$ iff $C \models$ choice(S1, letrec(p,call(p))) = choice(S2, letrec(p,call(p)))

(3) For every model N of CN there exists a model M of AN, such there is a partial homomorphism from N to M.

(4) For every model M of AN there exists a model N of CN such that there is a partial homomorphism from N to M.

(5) The type AN has an initial model I_A the restriction $I_A|_{STA}$ of which (to the constructor functions of statements) is isomorphic to the restriction $I|_{STA}$ of the initial model of CN.

Example: Let us consider the term S1 :

letrec(p, choice(nop, call(p)))

the term S2:

letrec(p,call(p))

and the term S3:

letrec(p, nop).

Then we have

$C \models$ loops(S1) = undefined,
$C \models$ loops(S2) = undefined,
$C \models$ loops(S3) = ff,
$C \models$ elem(S1,e,x) = (x ~ eval(e)),
$C \models$ elem(S2,e,x) = undefined,
$C \models$ elem(S3,e,x) = (x ~ eval(e)),

$B \models$ bloops(bn(S1)) = undefined,
$B \models$ bloops(bn(S2)) = undefined,
$B \models$ bloops(bn(S2)) = ff,
$B \models$ belem(bn(S1),e,x) = undefined,
$B \models$ belem(bn(S2),e,x) = undefined,
$B \models$ belem(bn(S3),e,x) = (x ~ eval(e)),

$A \models aloops(an(S1)) = ff,$
$A \models aloops(an(S2)) = undefined,$
$A \models aloops(an(S3)) = ff,$
$A \models elem(S1,e,x) = (x \sim eval(e)),$
$A \models elem(S2,e,x) = undefined,$
$A \models elem(S3,e,x) = (x \sim eval(e))$

According to this we have :

- S1, S2 and S3 are not visibly equivalent in C ,
- S1 and S2 are visibly equivalent in B ,
- S1 and S3 are visibly equivalent in A .

end of example

According to the axioms of BN we have for all nondeterministic statements S (we suppose that B and C have the same primitive models)

$$loops(S)^C = bloops(bn(S))^B ,$$
$$loops(S)^C = ff \Rightarrow elem(S,e,x) = belem(bn(S),e,c)$$
$$loops(S)^C = undefined \Rightarrow belem(S,e,x) = undefined$$

In particular wie have

$$belem(bn(S),e,x) \sqsubseteq elem(S,x,x)$$

where "$\sqsubseteq$" denotes Manna's "is less defined"-partial order (cf. /Manna 74/).

The reachably terminal models A and B of AN and BN resp. are incomparable.

The type LN , however, does not have initial nor weakly terminal models (cf. Fig.2):

Prop: (1) The type LN is not weakly sufficiently complete
(2) The type LN does not have any weakly terminal model nor any initial model
(3) For every model N of CN as well as of BN and AN there is a model M of LN which is isomorphic to A

For studying the relations between the models of LN we introduce the *implementation ordering* $\subseteq_I$ (cf. /Broy, Gnatz, Wirsing 78/):

$$L1 \subseteq_I L2 \quad \Leftrightarrow_{def}$$

for all lsta ls, exp e, dom x :

$$L2 \models lloops(ls) = ff \Rightarrow L1 \models lloops(ls) = ff$$
$$L1 \models lelem(ls,e,x) = tt \Rightarrow L2 \models lelem(ls,e,x) = ff$$

Then there does not exist unique minimum for $\subseteq_I$ in LN. But every $\subseteq_I$-minimal model L is a possible *deterministic mathematical semantics* for LN, i.e.

$$L \models (lelem(S,e,x) = tt \wedge x \neq y) \Rightarrow (lloop(S) = ff \wedge lelem(S,e,y) \neq tt)$$

Furthermore the weakly terminal model C of CN is *optimal* in the following sense: C is the weakest model of LN which is $\subseteq_I$-greater than all minimal models of LN; or equivalently C is the weakest model of LN which is $\subseteq_I$ - greater than all $\subseteq_I$-minimal models of LN (cf. figure 2).

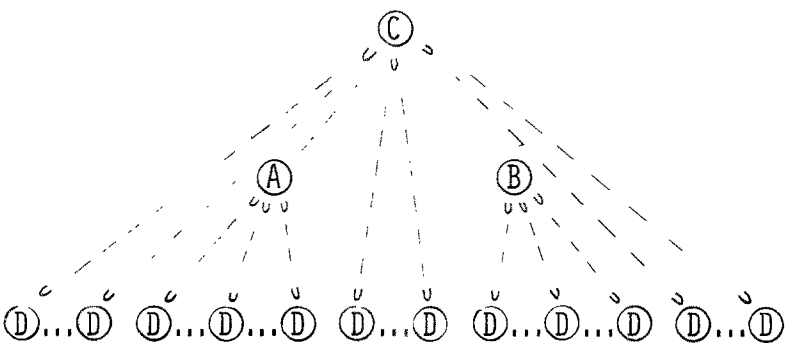

Figure 2: The implementation ordering $\subseteq_I$
A,B,C denote the classes of reachably terminal models of AN, BN and CN resp.
D denotes classes of extensionally equivalent, deterministic semantic models of LN.

Note: (1) In a deterministic mathematical semantics L for LN, i.e. in a $\subseteq_I$-minimal model L of type LN, we may introduce a partial function

$$\text{value} : \underline{\text{sta}} \times \underline{\text{exp}} \to \underline{\text{dom}}$$

such that

$$\text{value}(S,e) = x \text{ iff } L \models \text{lelem}(\text{ln}(S), e, x) = tt$$

(2) The meaning of the nondeterministic language of guarded commands in /Dijkstra 76/, which is very similar to our language, is defined by the predicate transformers of the wp-calculus. If we define the predicate calculus as primitive subtype with the sort predicate, the semantic function (cf. also "dynamic logic" in /Harel, Pratt 78/):

$$\text{wp} : \underline{\text{sta}} \times \underline{\text{predicate}} \to \underline{\text{predicate}}$$

defines backtrack nondeterminism (as weakly terminal model). An appropriate definition of the wlp-predicate transformers (cf. /Broy et al. 80/), however, leads to angelic nondeterminism, while the considering wp/wlp together gives choice nondeterminism.

This remark becomes obvious, if we define for our language:

$$\text{wlp}(S,R) = \lambda\, x\, .\, \forall\, \underline{\text{dom}}\ y : (\text{elem}(\text{semi}(\text{assign}(v,x),S),v,y) = tt \Rightarrow R(y))$$
$$\text{wp}(S,R) = \text{wlp}(S,R) \wedge \lambda\, x.\ (\text{loops}(\text{semi}(\text{assign}(v,x), S)) = ff)$$

where, for simplicity, we assume that v is the only ("generalized") program variable in the nondeterministic statement S and $x \in \underline{\text{dom}}$.

The axioms of BN and AN immediately give (cf./Broy et al. 80/):

$$BN \models \text{wlp}(S,R) = \text{wp}(S,R) \vee \neg\, \text{wp}(S,\underline{\text{true}})$$
$$AN \models \text{wp}(S,R) = \text{wlp}(S,R) \wedge \neg\, \text{wlp}(S,\underline{\text{false}})$$

end of note

We like to consider the minimally defined models of the types AN, BN, and CN resp., i.e. models which are extensionally equivalent to the reachably terminal models, as *tight* semantic models, whereas the models of LN (especially the deterministic ones) which are less than these models in the implementation ordering may be considered as *loose* semantic models for these types.

5. Concluding Remarks

The four different types properly reflect the four different notions of nondeterminism:

Backtrack nondeterminism assumes the computation of the "whole set of possible values". If there is a possibility of nontermination then this nontermination must happen. Thus backtrack nondeterminism is nothing but an implicit notation for programs working with sets. The choice is made *after* the computation of the set between the possible *semantic* values.

Choice nondeterminism represents a particular abstraction of a couple of decisions deliberately left open to the executing instance. Thus it corresponds to choices *during* the course of execution between alternative *statements* (i.e. the executing instance has the option of choice which statement to execute).

Unbounded nondeterminism corresponds to a "prophetic" choice during evaluation, avoiding nonterminating branches. Obviously we cannot give an operational semantics such that all possible values can be results, but nonterminating branches are excluded. This is reflected by the fact that unbounded nondeterminism is not continuous in the Egli-Milner ordering (cf. /Apt, Plotkin 81/). Nevertheless we may give approximations for operational semantics, i.e. models of type LN which are weaker than the partial initial model of AN.

Loose nondeterminism represents a convienient notation for treating a couple of possible semantic models in one specification. Thus it corresponds to choices *before* the execution of the program (or more understandable to choices of particular implementations, i.e. between *semantic models* , of a language). This comprises the choice of particular scheduling strategies in a compiler or operating system.

All four notions of nondeterminism have their justification in different areas of applications. *Angelic nondeterminism* is the notion used in automata theory. *Backtrack nondeterminism* can be used as a convenient notation for certain search problems (cf. /Floyd 67/). *Choice nondeterminism* serves as a formal basis for modelling concurrent processes (cf. /Broy 80/). Furthermore it can be used as a design tool for representing "program families", for expressing "delayed design decisions" (cf. /Bauer, Wössner 81/) or for explicit formulation of backtrack algorithms (cf. /Broy, Wirsing 80c/).

Of course, there are still other notions of nondeterminism. If we want to accept only specific objects (or situations) as possible results of computations this leads to a mixture of choice and backtrack nondeterminism assuming backtracking only in the specific situations ("exceptions").

References

/Apt, Plotkin 81/
K.R. Apt, G.D. Plotkin: A Cook's Tour of Countable Nondeterminism. Submitted for publication

/Bauer, Wössner 81/
F.L. Bauer, H. Wössner: Algorithmische Sprache und Programmentwicklung. Berlin-Heidelberg-New York: Springer 1981, to appear

/Broy 80/
M. Broy: Transformational Semantics for Concurrent Programs. IPL 11:2, October 1980, 87-91

/Broy, Gnatz, Wirsing 78/
M. Broy, R. Gnatz, M. Wirsing: Semantics of Nondeterministic and Noncontinuous Constructs. In: F.L. Bauer, M. Broy (eds.): Program Construction, Marktoberdorf 78. LNCS 69

/Broy, Wirsing 80a/
M. Broy, M. Wirsing: Programming Languages as Abstract Data Types. In: M. Dauchet (ed.): Lille Colloque 80

/Broy, Wirsing 80b/
M. Broy, M. Wirsing: Initial Versus Terminal Algebra Semantics for Partially Defined Abstract Types. Techn. Universität München, Institut für Informatik, TUM-I 8018, Dezember 1980

/Broy, Wirsing 80c/
M. Broy, M. Wirsing: From Enumeration to Backtracking. IPL 10:4, July 1980, 193-197

/Broy et al. 80/
M. Broy, H. Partsch, P. Pepper, M. Wirsing: Semantic Relations in Programming Languages, IFIP Congress 80

/Dijkstra 76/
E.W. Dijkstra: A Discipline of Programming. Prentice Hall, Englewood Cliffs 1976

/Floyd 67/
R.M. Floyd: Nondeterministic Algorithms. J. ACM 14, 1967, 636-644

/Grätzer 68/
G. Grätzer: Universal Algebra. Princeton: Van Nostrand 1968

/Harel, Pratt 78/
D. Harel, V.R. Pratt: Nondeterminism in Logics of Programs. Proc. 5th ACM Symp. on Principles of Programming Languages. Jan. 1978, 203-213

/Hennessy, Plotkin 80/
M.C.B. Hennessy, G.D. Plotkin: A Term Model of CCS. In: P. Dembinski(ed.): MFCS 80. LNCS 88, 262-274

/Kennaway, Hoare 80/
J.R. K. Kennaway, C.A.R. Hoare: A Theory of Nondeterminism. In: J. de Bakker, J.v.d. Leuwen (eds.): ICALP 80, LNCS 85

/Manna 74/
Z. Manna: Mathematical Theory of Computation. New York: McGraw Hill 1974

/McCarthy 63/
J. McCarthy: A Basis for a Theory of Computation. In: B. Bradfort, D. Hirschberg (eds.): Computer Programming and Formal Systems. Amsterdam: North-Holland 1963, 33-70

/Milner 77/
R. Milner: Fully Abstract Models of Typed λ-calculi. TCS 4, 1977, 1-22

/Nivat 80/
M. Nivat: Nondeterministic Programs: An Algebraic Overview. Invited paper, IFIP Congress 80

/Park 80/
D. Park: On the Semantics of Fair Parallelism. In: D. Björner (ed.): Abstract Software Specification. LNCS 86, 504-526

/Wirsing, Broy 80/
M. Wirsing, M. Broy: Abstract Data Types as Lattices of Finitely Generated Models. In: P. Dembinski (ed.): MFCS 80. LNCS 88