TEN YEARS OF HOARE'S LOGIC
A SURVEY - PART II: NONDETERMINISM

K.R. Apt
LITP, Paris, France

ABSTRACT

A survey of various results concerning the use of Hoare's logic in proving correctness of nondeterministic programs is presented. Various proof systems together with the example proofs are given and the corresponding soundness and completeness proofs of the systems are discussed. Programs allowing bounded and countable nondeterminism are studied. Proof systems deal with partial and total correctness, freedom of failure and the issue of fairness. The paper is a continuation of APT [1] where various results concerning Hoare's approach to proving correctness of sequential programs are presented.

1. INTRODUCTION

The purpose of this paper is to provide a systematic presentation of the use of Hoare's logic to prove correctness of nondeterministic programs. This paper is a continuation of APT [1] where we surveyed various results concerning the use of Hoare's logic in proving correctness of deterministic programs.

Hoare's method of proving programs correct was introduced in HOARE [14]. Even though it was originally proposed in a framework of sequential programs only, it soon turned out that the method can be perfectly well applied to other classes of programs, as well, in particular to the class of nondeterministic programs.

We discuss the issues in the framework of Dijkstra's nondeterministic programs introduced in DIJKSTRA [7] and concentrate on the issues of soundness and completeness of various proof systems.

This survey is divided into two parts dealing with bounded and countable nondeterminism in sections 3 and 4, respectively. A program allows bounded nondeterminism if at each moment in its execution at most a finite, fixed in advance number of possibilities can be pursued. If this number of possibilities can be countable then we say that the program allows countable nondeterminism.

In section 2 we introduce the basic definitions. In section 3 we discuss partial and total correctness of Dijkstra's programs. The methods used are straightforward generalizations of those which were introduced in the case of sequential programs and discussed in section 2 of APT [1]. This should be contrasted with the presentation in section 4 where total correctness of countably nondeterministic programs and total correctness of programs under the assumption of fairness is discussed. Even though the methods and techniques used there are appropriate generalizations of those used in section 3, various new insights are there needed. Finally, in section 5 bibliographical remarks are provided.

2. PRELIMINARIES

Throughout the paper we fix an arbitrary first order language L with equality containing two boolean constants true and false with obvious meaning. Its formulae are called *assertions* and denoted by letters p, q, r. Simple variables are denoted by letters a, b, x, y, z, expressions by letters s, t and quantifier-free formulae (*Boolean expressions*) by the letter e; p [t/x] stands for a *substitution* of t for all free occurrences of x in p.

All classes of programs considered in this paper contain the skip statement, the assignment statement x:=t and are closed under the composition of programs " ; ".

By a *correctness formula* we mean a construct of the form {p}S{q} where p, q are assertions and S is a program from a considered class. Correctness formulae are denoted by the letter ϕ.

An *interpretation* of L consists of a nonempty domain and assigns to each nonlogical symbol of L a relation or function over its domain of appropriate arity and kind. The letter J stands for an interpretation. Given an interpretation J by a *state* we mean a function assigning to all variables of L values from the domain of interpretation. States are denoted by letters σ, τ. The notions of a value of an expression t in a state σ (written as $\sigma(t)$) and truth of a formula p in a state σ (written as $\models_J p(\sigma)$) are defined in the usual way. A formula p is *true under* J (written as $\models_J p$) if $\models_J p(\sigma)$ holds for all states σ.

We allow two special states: $\bot$ reporting nontermination of a program and fail reporting a failure in execution of a program. We have by definition $\not\models_J p(\bot)$ $\not\models_J p$ (fail) for all formulae p. We define $[p]_J$ to the set all states σ which *satisfy* p *under* J (i.e. such that $\models_J p(\sigma)$ holds). Thus by definition for any p and J $\bot \notin [p]_J$ and fail $\notin [p]_J$.

Finally, let Tr_J be the set of all assertions which are true under J.

3. BOUNDED NONDETERMINISM

Denote by S_n the least class of programs such that for all boolean expressions $e_1, \ldots e_m$ and $S_1, \ldots, S_m \in S_n$ $\underline{if}$ $e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m$ $\underline{fi} \in S_n$ and $\underline{do}$ $e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m$ $\underline{od} \in S_n$.

This class of programs was introduced in DIJKSTRA [7] and further extensively studied in DIJKSTRA [8] and various other papers. The boolean expressions e_i in the context of the if and do - constructs are called *guards*. An intuitive meaning of the program if $e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m$ fi is: choose nondeterministically a guard e_i which evaluates to true and execute the program S_i. In the case when all guards $e_1, \ldots, e_m$ evaluate to false the program *fails*, i.e. its execution improperly terminates. An intuitive meaning of the program do $e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m$ od is : as long as at least one guard evaluates to true repeatedly do the following: choose any guard e_i which evaluates to true and execute the program S_i. In the case of one guard only the construct do $e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m$ od is thus equivalent to the usual construct while e_1 do S_1 od.

3.1. Semantics of nondeterministic programs

Before we dwell on the issue of correctness of the programs from S_n we define their semantics. We follow here the approach of HENNESSY & PLOTKIN [13] the advantage of which is that it can be easily adopted to several other classes of programs. This semantics is based on the consideration of a transition relation "$\rightarrow$" between pairs $\langle S, \sigma\rangle$ consisting of a program S and a state σ. The intuitive meaning of the relation

$$\langle S_1, \sigma\rangle \rightarrow \langle S_2, \tau\rangle$$

is: executing S_1 one step in a state σ can lead (nondeterministically) to a state τ with S_2 being remainder of S_1 still to be executed. It is convenient to assume the empty program E. Then S_2 is E if S_1 terminates in τ. We assume that for any S E;S = S;E = S.

Given an interpretation we define the above relation by the following clauses:

(i) $<\underline{skip}, \sigma> \rightarrow <E,\sigma>$

(ii) $<x:=t, \sigma> \rightarrow <E,\tau>$

where $\tau(x) = \sigma(t)$ and $\tau(y) = \sigma(y)$ for $y \not\equiv x$

(iii) $<\underline{if}\ e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m \underline{fi}, \sigma> \rightarrow <S_i,\sigma>$ if $\models_J e_i(\sigma)$

(iv) $<\underline{if}\ e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m \underline{fi}, \sigma> \rightarrow <E, \underline{fail}>$ if $\models_J \bigwedge_{j=1}^{m} \neg e_i(\sigma)$

(v) $<\underline{do}\ e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m \underline{od}, \sigma> \rightarrow <S_i ; \underline{do}\ e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m \underline{od},\sigma>$
$\underline{if}$ $\models_J e_i(\sigma)$

(vi) $<\underline{do}\ e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m \underline{od}, \sigma> \rightarrow <E, \sigma>$ if $\models_J \bigwedge_{i=1}^{m} \neg e_i(\sigma)$

(vii) if $<S_1,\sigma> \rightarrow <S_2,\tau>$ then $<S_1 ; S, \sigma> \rightarrow <S_2 ; S, \tau>$.

Let $\rightarrow^*$ stand for the transitive, reflexive closure of $\rightarrow$.

We now introduce the following definitions.

DEFINITION

(i) S can *diverge from* σ if there exists an infinite sequence $<S_i,\sigma_i>$ $(i = 0,1,\ldots)$ such that $<S,\sigma> = <S_0,\sigma_0> \rightarrow <S_1,\sigma_1> \rightarrow \ldots$

(ii) S can *fail from* σ if $<S,\sigma> \rightarrow^* <S_1,\underline{fail}>$ for some S_1.

(iii) A finite sequence $<S_i,\sigma_i>$ $(i=0,1,\ldots,k)$ such that $<S,\sigma> = <S,\sigma_0> \rightarrow <S_1,\sigma_1> \rightarrow \ldots \rightarrow <S_k,\sigma_k> = <E,\sigma_k>$ is called a *computation starting in* $<S,\sigma>$; k is the length of this computation.

The following lemma will be needed later.

LEMMA 1. *If* S *cannot diverge from* σ *then there exists a natural number* k *such that all computations starting in* $<S,\sigma>$ *are of length at most* k.

PROOF. Consider the set of all finite sequences $<S,\sigma> = <S_0,\sigma_0> \rightarrow \ldots \rightarrow <S_n,\sigma_n>$ ordered by the subsequence ordering. This set forms a finitely branching tree. If the desired k did not exist then this tree would be infinite. By König's Lemma it would then contain an infinite branch which contradicts the assumption. $\square$

We define now two types of semantics for the programs from S_n by putting

$$M[\![S]\!](\sigma) = \{\tau \mid \langle S,\sigma\rangle \rightarrow^* \langle E,\tau\rangle\}$$

and

$$M_{tot}[\![S]\!](\sigma) = M[\![S]\!](\sigma) \;\cup\; \{\bot \mid S \text{ can diverge from } \sigma\} \;\cup\; \{\underline{fail} \mid S \text{ can fail from } \sigma\}$$

Both semantics depend on the interpretation J but we do not mention this dependence hoping that no confusion will arise. The difference between these two semantics lies in the way the "negative" informations about the program are dealt with - either they are dropped or they are explicitly mentioned.

3.2. Partial and total correctness

While studying a correctness of programs we are interested in various properties namely

(a) whether all proper states generated (or produced) by the program satisfy a given post-condition,

(b) whether the program always terminates, and

(c) whether none of the executions of the program leads to a failure.

We are usually interested in executions starting in a state satisfying some initial pre-condition. The above properties lead to various possible interpretations of the correctness formulae {p} S {q}. Let

$$M[\![S]\!]([p]_J) = \bigcup_{\sigma\in[p]_J} M[\![S]\!](\sigma)$$

and

$$M_{tot}[\![S]\!]([p]_J) = \bigcup_{\sigma\in[p]_J} M_{tot}[\![S]\!](\sigma).$$

We define

$$\models_J \{p\}\, S\, \{q\} \text{ iff } M[\![S]\!]([p]_J) \subseteq [q]_J,$$

$$\models_{J,\, tot} \{p\}\, S\, \{q\} \text{ iff } M_{tot}[\![S]\!]([p]_J) \subseteq [q]_J.$$

Informally speaking, $\models_J$ {p} S {q} means that any properly terminating execution of S starting in a state satisfying p leads to a state satisfying q ;

$\models_{J,tot}$ {p} S {q} in addition guarantees that any execution of S starting in a state satisfying p properly terminates. If $\models_J$ {p} S {q} holds we say that the program S is *partially correct under* J (with respect to p and q). If $\models_{J,tot}$ {p} S {q} holds we say that the program S is *totally correct under* J (with respect to p and q).

3.3. A proof system for partial correctness

We now present a formal system allowing us to deduce formally partial correctness of programs from $\mathcal{S}_n$. Its axioms and proof rules are the following

AXIOM 1 : skip axiom

{p} skip {p}

AXIOM 2 : assignment axiom

{p[t/x]} x:=t {p}

RULE 3 : composition rule

$$\frac{\{p\}\ S_1\{r\},\{r\}\ S_2\{q\}}{\{p\}\ S_1\ ;\ S_2\ \{q\}}$$

RULE 4 : if - rule

$$\frac{\{p \wedge e_i\}\ S_i\ \{q\},\ i = 1,\dots,m}{\{p\}\ \underline{\text{if}}\ e_1 \rightarrow S_1\ \Box\ \dots\ \Box\ e_m \rightarrow S_m\ \underline{\text{fi}}\{q\}}$$

RULE 5 : do - rule

$$\frac{\{p \wedge e_i\}\ S_i\ \{p\},\ i = 1,\dots,m}{\{p\}\ \underline{\text{do}}\ e_1 \rightarrow S_1\ \Box\ \dots\ \Box e_m \rightarrow S_m\ \underline{\text{od}}\ \{p \wedge \bigwedge_{i=1}^{m} \neg e_i\}}$$

p is called a *loop invariant*.

RULE 6 : consequence rule

$$\frac{p \rightarrow p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \rightarrow q}{\{p\}\ S\ \{q\}}$$

We call this proof system N. For A being a set of assertions and a correctness formula ϕ we write $A \vdash_N \phi$ to denote the fact that there exists a proof of ϕ in N which uses as assumptions for the consequence rule assertions from A.

3.4. An example of a proof in N

To illustrate the use of the proof system N we now provide the following example. Let S stand for the following program

```
do 2|x ∨ 3|x →
   if 2|x → x:=x/2 ; a:=b+1
     □3|x → x:=x/3 ; b:=b+1
     □4|x → x:=x/4 ; a:=a+2 fi
od
```

where x, a, b are integer variables. This program computes the greatest powers of 2 and 3 which divide x. We now present a formal proof of this fact. More precisely we prove

(1) $$Tr_{J_0} \models_{N} \{a = 0 \wedge b = 0 \wedge x = z\}\ S\ \{z = x.2^a.3^b \wedge \neg(2|x \vee 3|x)\}$$

where J_0 is the standard interpretation of the language of Peano arithmetic augmented with the division operation and divisibility relation.

We present the proof in a "top-down" fashion. We choose $p \equiv z = x\cdot 2^a\cdot 3^b$ to be the loop invariant. We now show

(2) $$a = 0 \wedge b = 0 \wedge x = z \rightarrow p,$$

(3) $$\{p \wedge (2|x \vee 3|x)\}\ S_1\ \{p\}$$

where S_1 is the loop body,

(4) $$p \wedge\neg (2|x \vee 3|x) \rightarrow z = x\cdot 2^a\cdot 3^b \wedge\neg (2|x \vee\neg (2|x \vee 3|x).$$

Note that (3) implies by the do-rule $\{p\}\ S\ \{p \wedge\neg (2|x \vee 3|x)\}$ which together with (2) and (4) implies by the consequence rule (1). Both (2) and (4) are obvious.

To show (3) we have to show

(5) $\{p \wedge (2|x \vee 3|x) \wedge 2|x\}$ x:=x/2 ; a:=a+1 $\{p\}$,

(6) $\{p \wedge (2|x \vee 3|x) \wedge 3|x\}$ x:=x/3 ; b:=b+1 $\{p\}$,

(7) $\{p \wedge (2|x \vee 3|x) \wedge 4|x\}$ x:=x/4 ; a:=a+2 $\{p\}$

and apply the if-rule.

We now prove (5). By the assignment axiom

$$\{z = x \cdot 2^{a+1} \cdot 3^b\} \text{ a:=a+1 } \{p\}$$

and

$$\{z = (x/2) \cdot 2^{a+1} \cdot 3^b\} \text{ x:=x/2 } \{z = x \cdot 2^{a+1} \cdot 3^b\}$$

so by the composition rule $\{z = (x/2) \cdot 2^{a+1} \cdot 3^b\}$ x:=x/2 ; a:=a+1 $\{p\}$ which by the consequence rule implies (5). Proofs of (6) and (7) are similar and left to the reader.

Note. To ensure that the application of the division operation does not result in producing non-integer values we should actually use here the following assignment rule in the case of division operation:

$$\frac{p[(a/b)/x] \rightarrow b|a}{\{p[(a/b)/x]\text{x := a/b}\{p\}} \; .$$

We leave it to the reader checking that the above proof remains correct when this assignment rule is used.

3.5. Soundness of N

To justify the proofs in the system N one has to prove its *soundness* in the sense of the following theorem which links provability of the correctness formulae with their truth.

THEOREM 1. *For every interpretation* J, *set of assertions* A *and correctness formula* ϕ *the following holds: if all assertions from* A *are true under* J *and* $A \vdash_N \phi$ *then* ϕ *is true under* J.

In other words if $Tr_J \vdash_N \phi$ *then* $\models_J \phi$.

We call correctness formula *valid* if it is true under all interpretations J and a proof rule *sound* if for all interpretations J it preserves the truth under J of correctness formulae (and in the case of the consequence rule, assertions).

To prove the soundness of N it is sufficient to show that all axioms of N are valid and all proof rules of N are sound since the desired conclusion follows then by the induction on the length of proofs. As an example proof we now show the soundness of the do-rule.

Let S stand for $\underline{do}\ e_1 \rightarrow S_1 \square \ldots \square e_m \rightarrow S_m \underline{od}$. Fix an interpretation J and assume that all the premises of the do-rule are true under J, i.e. that

(8) $\quad M [\![S_i]\!] ([p \wedge e_i]_J) \subseteq [p]_J$ for $i = 1,\ldots,m$.

Let $\tau \in M [\![S]\!] ([p]_J)$. Then for some $\sigma \in [p]_J$ $\tau \in M [\![S]\!] (\sigma)$. By the definition of M we have

$$<S,\sigma_0> \rightarrow^* <S_1,\sigma_1> \rightarrow^* \ldots \rightarrow^* <S_\ell,\sigma_\ell> \rightarrow <E,\sigma_\ell>$$

where $\sigma = \sigma_0$, $\tau = \sigma_\ell$ and for all $j = 0,\ldots, \ell - 1$ $\sigma_j \in [e_{k_j}]_J$ and $\sigma_{j+1} \in M [\![S_{k_j}]\!] (\sigma_j)$ for some $k_j \in \{1,\ldots,m\}$ and $\sigma_\ell \in [\bigwedge_{i=1}^{m} \neg e_i]_J$. We have $\sigma_0 \in [p]_J$ and if for some $j \in \{0,\ldots,\ell-1\}$ $\sigma_j \in [p]_J$ then by (8) $\sigma_{j+1} \in M [\![S_{k_j}]\!] ([p \wedge e_{k_j}]_J) \subseteq [p]_J$, i.e. $\sigma_{j+1} \in [p]_j$. Thus for all $j = 0,\ldots,\ell$ $\sigma_j \in [p]_J$. In particular $\sigma_\ell \in [p]_J$ which means that $\tau \in [p \wedge \bigwedge_{i=1}^{m} \neg e_i]_J$. This proves the truth under J of the conclusion of the do-rule and thereby concludes the proof of the soundness of the rule.

3.6. Completeness of N in the sense of Cook

A converse property to that of soundness of a proof system is completeness which links truth of the correctness formulae with their provability. Unfortunately a converse implication to this theorem 1 can be proved only for a special type of interpretations J. This issue is discussed at length in APT [1] in sections 2.7. and 2.8. where we refer the reader for the details. We restrict ourselves here to presenting the appropriately adopted definitions without entering into any discussion of the results.

Define

$$\mathrm{post}_J(p,S) = \mathcal{M}[\![S]\!]([p]_J)$$

$$\mathrm{pre}_J\ (S,q) = \{\sigma : \mathcal{M}[\![S]\!](\sigma) \subseteq [q]_J\}$$

Note that these sets are characterized by the following equivalences (the second of them is just a rewording of the definition):

$$(9)\qquad \models_J\{p\}\ S\ \{q\} \quad \text{iff} \quad [p]_J \subseteq \mathrm{pre}_J(S,q) \quad \text{iff} \quad \mathrm{post}_J(p,S) \subseteq [q]_J.$$

Let $\mathcal{S}_0$ be a class of programs.

Call the language L *expressive relative to* J *and* $\mathcal{S}_0$ if for all assertions p and programs $S \in \mathcal{S}_0$ there exists an assertion q which defines $\mathrm{post}_J(p,S)$. If J is such that L is expressive relative to J and $\mathcal{S}_0$ we write $J \in \mathrm{Exp}(L,\mathcal{S}_0)$. It is worthwhile to note that in the definition of expressiveness we can alternatively require definability of $\mathrm{pre}_J(S,q)$ instead of $\mathrm{post}_J(p,S)$ (see APT [1]).

Definition A proof system G for S_0 is complete *in the sense of Cook* if, for every interpretation $J \in \mathrm{Exp}(L,\mathcal{S}_0)$ and every asserted program ϕ if $\models_J\phi$, then $\mathrm{Tr}_J \vdash_G \phi$.

This definition of completeness is, as the name indicates, due to COOK [6].

Now, the proof system N for $\mathcal{S}_n$ is complete in the sense of Cook. The proof proceeds by induction on the structure of the programs.

The only two nontrivial cases are these of composition and the do-construct.

If $\models_J\{p\}\ S_1;S_2\ \{q\}$ then clearly $\models_J\{p\}\ S_1\ \{r\}$ and $\models_J\{r\}S_2\{q\}$ where r defines $\mathrm{pre}_J(S_2,q)$; so, by the induction hypothesis and the composition rule, $\mathrm{Tr}_J \vdash_N \{p\}S_1 ; S_2\{q\}$. If $\models_J\{p\}S\{q\}$, where $S \equiv$ do $e_1 \to S_1 \square \ldots \square e_m \to S_m$ od, then we must find a loop invariant r such that for $i = 1,\ldots,m$ $\models_J\{r \wedge e_i\}\ S_i\ \{r\}$, $\models_J p \to r$ and $\models_J(r \wedge \bigwedge_{i=1}^{m} \neg e_i) \to q$. Then by the induction hypothesis and the consequence rule $\mathrm{Tr}_J \vdash_N \{p\}\ S\ \{q\}$.

We choose r to be an assertion defining $\mathrm{pre}_J(S,q)$. Then by (9) $\models_J\{r\}\ S\ \{q\}$ so also $\models_J\{r\}$ if $e_i \to S_i \square \neg e_i \to$ skip fi ; $S\{q\}$ for all

$i = 1,\ldots,m$ as for any σ $M[\![\underline{if}\ e_i \to S_i \,\square\, \neg e_i \to \underline{skip}\ \underline{fi}\ ;\ S]\!](\sigma) \subset M[\![S]\!](\sigma)$ clearly holds. Now, since r defines $pre_J(S,q)$, then as in the case treated above $\models_J \{r\}\ \underline{if}\ e_i \to S_i \,\square\, e_i \to \underline{skip}\ \underline{fi}\ \{r\}$ from which $\models_J \{r \wedge e_i\}\ S_i\ \{r\}$ follows. By (9) we have $\models_J p \to r$ and $\models_J (r \wedge \bigwedge_{i=1}^{m} \neg e_i) \to q$ follows from the definition of r. This concludes the proof.

3.7 A proof system for total correctness

To prove total correctness of programs from S_n we must provide proof rules ruling out possibility of failure and nontermination.

A possible failure in an execution of a program from S_n can be caused only by the if-construct. Clearly the if-rule does not rule out a possibility of failure. However, a small refinement of this rule suffices to prove the lack of failure. We only need to ensure that at each moment an if-statement is to be executed at least one of its guards evaluates to true. This is achieved by the following modification

RULE 7 : if-rule II

$$\frac{p \to \bigvee_{i=1}^{m} e_i,\ \{p \wedge e_i\}\ S_i\ \{q\}_{i=1,\ldots,m}}{\{p\}\ \underline{if}\ e_1 \to S_1 \,\square \ldots \square\, e_m \to S_m\ \underline{fi}\ \{q\}}$$

A possible nontermination of an execution of a program from S_n can be caused only by the do-construct and clearly the present do-rule does not rule out such a possibility. The following modification of the do-rule suffices to prove termination of each do-construct. This rule is due to HAREL [11] where a different formalism is used.

RULE 8 : do-rule II

$$\frac{\begin{array}{c} p(n) \wedge n > 0 \to \bigvee_{i=1}^{m} e_i\ ,\ p(0) \to \bigwedge_{i=1}^{m} \neg e_i, \\ \{p(n) \wedge n > 0 \wedge e_i\}\ S_i\ \{\exists m < n\ p(m)\}_{i=1,\ldots,m} \end{array}}{\{\exists n\ p(n)\}\ \underline{do}\ e_1 \to S_1 \,\square \ldots \square\, e_m \to S_m\ \underline{od}\ \{p(0)\}}$$

Here p(n) is an assertion with a free variable n which does not appear in the programs and ranges over natural numbers.

Let NT denote the proof system obtained from N by replacing the if and do-rules by their modified versions. This proof system is appropriate for proving total correctness of programs from S_n.

To illustrate the use of the system we now indicate how to modify the proof given in section 3.4. to demonstrate the total correctness of the program there considered, i.e. to prove (1) within NT.

We choose $p(n) \equiv p \wedge \exists a_1,b_1,x_1\ (x=2^{a_1}\cdot 3^{b_1}\cdot x_1 \wedge \neg(2|x_1 \vee 3|x_1) \wedge n=a_1+b_1)$.

The second component of p(n) states that n is the sum of powers of 2 and 3 which divide x.

We now have

(10) $\quad a = 0 \wedge b = 0 \wedge x = z \rightarrow \exists n\ p(n)$,

(11) $\quad p(n) \wedge n > 0 \rightarrow 2|x \vee 3|x$,

(12) $\quad p(0) \rightarrow \neg(2|x \vee 3|x)$,

(13) $\quad \{p(n) \wedge n > 0\}\ S_1\ \{\exists m < n\ p(m)\}$

where the last correctness formula can be proved using the <u>if</u>-rule II since $p(n) \wedge n > 0 \rightarrow 2|x \vee 3|x \vee 4|x$ holds. The proof of (13) is a small modification of the proof of (3) and is left to the reader. Now by the <u>do</u>-rule II, (10) and (12) we obtain (1) as desired.

3.8. Arithmetical soundness and completeness of NT

As explained in section 2.11 of APT [1] when trying to prove soundness of a proof for total correctness one has to revise appropriately the notion of soundness. We follow here the approach of HAREL [11] also adopted in APT [1]. We recall the introduced definitions.

Let L be an assertion language and let L^+ be the minimal extension of L containing the language L_p of Peano arithmetic and a unary relation nat(x). Call an interpretation J of L^+ *arithmetical* if its domain includes the set of natural numbers, J provides the standard interpretation for L_p, and nat(x), is interpreted as the relation "to be a natural number". Additionally, we require that there exists a formula of L^+ which, when interpreted under J, provides the ability to encode finite sequences of elements from the domain of J into on element. (The last requirement is needed only for the completeness proof.)

One of the examples of an arithmetical interpretation is of course J_0. It is important to note that any interpretation of an assertion language L with an infinite domain can be extended to an arithmetical interpretation of L^+. Clearly, the proof system NT is suitable only for assertion languages of the form L^+, and an expression such as p(n) is actually a shorthand for nat(n) $\wedge$ p(n).

We now say that a proof system G for total correctness is *arithmetically sound* if, for all arithmetical interpretations J and asserted programs ϕ $Tr_J \vdash_G \phi$ implies $\models_{J,tot} \phi$.

It can be shown that the proof system NT is arithmetically sound. The case of the <u>if</u>-rule II is easily handled. The proof of soundness of the <u>do</u>-rule II for the case of arithmetical interpretations is in turn an easy modification of the proof of soundness of the <u>do</u>-rule where one simply parametrizes the invariant p. The proofs of other cases are the same as before.

We say that a proof system G is *arithmetically complete* if for all arithmetical interpretations J and asserted programs ϕ $\models_{J,tot} \phi$ implies $Tr_J \vdash_G \phi$.

To show the arithmetical completeness of the system NT we first introduce the following notion:

$$pret_J(S,q) = \{\sigma : M_{tot}[\![S]\!](\sigma) \subseteq [q]_J\}.$$

pret stands in the same relation to total correctness as pre does to partial correctness : we have $\models_{J,tot} \{p\}\ S\ \{q\}$ iff $[p]_J \subseteq pret_J(S,q)$.

Thanks to the provision for coding of finite sequences it can be shown that for any arithmetical interpretation J there exists an assertion which defines $pret_J(S,q)$. This fact is not completely obvious as the definition of $pret_J(S,q)$ also mentions (the nonexistence of) infinite sequences. This difficulty can be however circumvented by making use of Lemma 1.

The completeness proof proceeds by induction on the structure of programs. The only cases different from the corresponding ones in the completeness proof of N are those of <u>if</u> and <u>do</u>-constructs. Let J be an arithmetical interpretation.

If $\models_{J,tot} \{p\}$ <u>if</u> $e_1 \rightarrow S_1 \,\Box \ldots \Box e_m \rightarrow S_m$ <u>fi</u> $\{q\}$ then by definition $\models_J p \rightarrow \bigvee_{i=1}^{m} e_i$ and $\models_{J,tot} \{p \wedge e_i\}\ S_i\ \{q\}$ for $i = 1,\ldots,m$. By the induction hypothesis $Tr_J \vdash_{NT} \{p \wedge e_i\}\ S_i\ \{q\}$ for $i = 1,\ldots,m$ so by the <u>if</u>-rule II $Tr_J \vdash_{NT} \{p\}$ <u>if</u> $e_1 \rightarrow S_1 \,\Box \ldots \Box e_m \rightarrow S_m$ <u>fi</u> $\{q\}$.

Assume now $\models_{J,tot} \{r\}\ S\{q\}$ where $S \equiv$ <u>do</u> $e_1 \rightarrow S_1 \,\Box \ldots \Box e_m \rightarrow S_m$ <u>od</u>. Let n be a fresh variable. Let now C be the following set of states:

$$\mathrm{pret}_J(S,q) \cap \{\sigma : \models_J \mathrm{nat}(n)(\sigma) \wedge \text{the longest computation starting in } \langle S,\sigma\rangle \text{ is of length } k+1, \text{ where } k = \sigma(n)\}.$$

Thus $\sigma \in C$ iff $\sigma(n)$ is a natural number, say k, such that all computations starting in $\langle S,\sigma\rangle$ properly terminate in a state satisfying q and the longest of these computations is of length k+1. It can be shown that there exists an assertion p(n) which defines C.

By the definition of p(n) we now have $\models_J p(n) \wedge n > 0 \rightarrow \bigvee_{i=1}^{m} e_i$, $\models_J p(0) \rightarrow \bigwedge_{i=1}^{m} \neg e_i$. Also it can be easily shown that $\models_J \{p(n) \wedge n > 0 \wedge e_i\}\ S_i\ \{\exists m < n\ p(m)\}$. By the induction hypothesis and the <u>do</u>-rule II we get $Tr_J \vdash_{\overline{NT}} \{\exists n\ p(n)\}$ <u>do</u> $e_1 \rightarrow S_1 \,\square\, \ldots \,\square\, e_m \rightarrow S_m$ <u>od</u> $\{p(0)\}$.

We now have by assumption $[r]_J \subseteq \mathrm{pret}_J(S,q)$ and so by virture of Lemma 1 $\models_J r \rightarrow \exists n\ p(n)$. Also $\models_J p(0) \rightarrow q$ holds so by the consequence rule we get $Tr_J \vdash_{\overline{NT}} \{r\}$ <u>do</u> $e_1 \rightarrow S_1 \,\square\, \ldots \,\square\, e_m \rightarrow S_m$ <u>od</u> $\{q\}$.

This concludes the proof.

4. COUNTABLE NONDETERMINISM

<u>4.1. Bounded nondeterminism versus finite and countable nondeterminism</u>

Up till now we have considered programs which allowed *bounded nondeterminism* only. By this we mean that for each pair $\langle S,\sigma\rangle$ where $S \in \mathcal{S}_n$ the set $\{\langle S_1, \sigma_1\rangle : \langle S, \sigma\rangle \rightarrow \langle S_1, \sigma_1\rangle\}$ is finite and moreover its cardinality is *bounded* by a constant dependent on S only. Informally it means that each program $S \in \mathcal{S}_n$ gives rise in one computation step to at most k different continuations where k depends on S only.

This property should be contrasted with that of *finite nondeterminism* which means that the above set is always finite but its cardinality does not depend on S only. An example of an instruction which leads to finite nondeterminism is $x:=? \leq y$ which sets to x a value smaller or equal to y. Such an instruction has been considered in FLOYD [9]. (Of course, we assume here that the programs are interpreted under a standard interpretation in natural numbers.)

It should be however noted that finite nondeterminism can be reduced to a bounded nondeterminism in the sense that $x:=? \leq y$ is equivalent to a program from $\mathcal{S}_n$. To see this take for example the program b:=<u>true</u>; x:=0 ; <u>do</u> $b \wedge x < y \rightarrow x:=x+1 \,\square\, b \wedge x < y \rightarrow$ b:=<u>false</u> <u>od</u>. Consequently the study of finite nondeterminism (in the above sense) can be reduced to the study of bounded nondeterminism.

This is not any more the case with *countable nondeterminism*. By countable nondeterminism we mean that the above defined set can be countably infinite. An example of an instruction which leads to countable nondeterminism is the *random assigment* x:=? which sets to x an arbitrary nonnegative integer.

It is obvious how to define the semantics $M_{tot}[\![x:=?]\!]$ of x:=?. We have $\perp \notin M_{tot}[\![x:=?]\!](\sigma)$ for any σ. We now claim that there is no program $S \in S_n$ such that $M_{tot}[\![x:=?]\!] = M_{tot}[\![S]\!]$. This follows immediately from the following corollary to Lemma 1.

Corollary 1. *For any* $S \in S_n$ *and* σ *if* $\perp \notin M_{tot}[\![S]\!](\sigma)$ *then* $M_{tot}[\![S]\!](\sigma)$ *is a finite set.* □

Thus countable nondeterminism cannot be reduced to bounded (or finite) nondeterminism. This indicates that to study total correctness of programs allowing countable nondeterminism we have to develop essentially new proof rules, i.e. proof rules which cannot be derived from those of the proof system NT.

Note that this is not the case when dealing with the partial correctness of programs allowing countable nondeterminism as clearly

$$M[\![x:=?]\!] = M[\![b:=\underline{true}\ ;\ x:=0\ ;\ \underline{do}\ b \rightarrow x:=x+1\ \square\ b \rightarrow b:=\underline{false}\ \underline{od}]\!]$$

(In this and the above considerations we ignored the fact that the value of b has been changed. It is easy to remedy this problem.)

Before we enter the proof theoretic considerations of countable nondeterminism we should perhaps explain why it is useful to study countable nondeterminism in the first place. First, the instruction x:=? can be viewed as another version of a more familiar read (x) instruction. Secondly, this instruction is particularly useful when dealing with the assumption of *fairness*, which will be discussed later. Also it allows to provide various neat characterizations of objects discussed in mathematical logic (see e.g. HAREL & KOZEN [12]).

4.2. A proof system for total correctness of countably nondeterministic programs

Consider now the class S_{cn} of programs which differs from S_n in that additionnally the instruction x:=? is allowed. We now present a proof system which allows us to prove total correctness of programs from S_{cn}. We add to the proof system NT the following axiom

AXIOM 9: random assignment axiom

$\{p\}x:=?\ \{p\}$

provided x is not free in p

and replace the do-rule II by the following generalization of it:

RULE 10: do-rule III

$$\frac{p(\alpha) \wedge \alpha > 0 \rightarrow \bigvee_{i=1}^{m} e_i,\ p(0) \rightarrow \bigwedge_{i=1}^{m} \neg e_i,\quad \{p(\alpha) \wedge \alpha > 0 \wedge e_i\}\ S_i\ \{\exists\beta < \alpha\ p(\beta)\},\ i = 1,\dots,m}{\{\exists\alpha\ p(\alpha)\}\ \underline{do}\ e_1 \rightarrow S_1 \square \dots \square e_m \rightarrow S_m\ \underline{od}\ \{p(0)\}}$$

where p(α) is an assertion with a free variable α which does not appear in the programs and ranges over ordinals.

Call the resulting proof system CNT.

4.3. An example of a proof in CNT

As an example proof in CNT consider now the following program:

$$S \equiv \underline{do}\ x=0 \rightarrow y:=?\ ;\ x:=1$$
$$\square\ x \neq 0 \wedge y > 0 \rightarrow y:=y-1$$
$$\underline{do}.$$

We now wish to prove in CNT that S always terminates. More precisely, we prove in CNT the correctness formula {true} S {y=0}.

To this end we first specify the assertion language L. We assume that L contains the language of Peano arithmetic and has two sorts: data (for program data - here integer) and ord for ordinals. We assume a constant 0 of sort ord and a binary predicate symbol < over ord. The variables α, β are of sort ord, all other variables are of sort data.

In the course of the proof we shall have to convert values of sort data into values of sort ord. To this purpose we assume a one-argument conversion function $\bar{\cdot}$ of sort (data, ord) converting integers into ordinals and a constant ω of sort ord. We have $\forall x\ (\bar{x} < \omega)$ as by convention x is of type data.

Define $p(\alpha)$ by

$$p(\alpha) \equiv (x = 0 \rightarrow \alpha = \omega) \wedge (x \neq 0 \rightarrow \alpha = \bar{y}).$$

Intuitively speaking, for a state σ, $p(\alpha)(\sigma)$ holds if α is the smallest ordinal greater than or equal to the number of possible iterations performed by the loop when started in σ.

We now show that $p(\alpha)$ satisfies the premises of the do-rule III, i.e. that $p(\alpha)$ is a loop invariant.

1. We have $p(\alpha) \wedge \alpha > 0 \rightarrow x = 0 \vee y > 0 \rightarrow x = 0 \vee (x \neq 0 \wedge y > 0)$
2. We have $p(0) \rightarrow x \neq 0 \wedge y = 0 \rightarrow \neg(x = 0 \vee (x \neq 0 \wedge y > 0))$
3. We first show $\{p(\alpha) \wedge \alpha > 0 \wedge x = 0\}$ y:=? ; x:=1 $\{\exists\beta < \alpha\ p(\beta)\}$.

By the assignment axiom we have

$$\{\exists\beta < \alpha\ p(\beta)\ [1/x]\}\ x:=1\ \{\exists\beta < \alpha\ p(\beta)\}$$

so by the consequence rule

$$\{\forall y\ \exists\beta < \alpha\ p(\beta)[1/x]\}\ x:=1\ \{\exists\beta < \alpha\ p(\beta)\}.$$

By the random assignment axiom and the composition rule we now get

$$\{\forall y\ \exists\beta < \alpha\ p(\beta)\ [1/x]\}\ y:=?\ ;\ x:=1\ \{\exists\beta < \alpha\ p(\beta)\}$$

To complete the proof it now suffices to show that $p(\alpha) \wedge \alpha > 0 \wedge x = 0 \rightarrow \forall y\ \exists\beta < \alpha\ p(\beta)\ [1/x]$ is true. $p(\alpha) \wedge x = 0$ implies $\alpha = \omega$. So for any y put $\beta = \bar{y}$: then $\beta < \alpha$ and $p(\beta)[1/x]$ holds.

Next we show $\{p(\alpha) \wedge \alpha > 0 \wedge x \neq 0 \wedge y > 0\}$ y:=y-1 $\{\exists\beta < \alpha\ p(\beta)\}$. By the assignment axiom and the consequence rule it suffices to show that $p(\alpha) \wedge \alpha > 0 \wedge x \neq 0 \wedge y > 0 \rightarrow \exists\beta < \alpha\ p(\beta)\ [y-1/y]$ is true. We have

$$\begin{aligned} p(\alpha) \wedge \alpha > 0 \wedge x \neq 0 \wedge y > 0 &\rightarrow \alpha = \bar{y} \wedge y > 0 \wedge x \neq 0 \\ &\rightarrow \alpha = \bar{y} \wedge y > 0 \wedge p(\overline{y-1})[y-1/y] \\ &\rightarrow \exists\beta < \alpha\ p(\beta)[y-1/y] \end{aligned}$$

By the do-rule III we now get

$$\{\exists\alpha\ p(\alpha)\}\ S\ \{p(0)\}.$$

Clearly both $\exists\alpha\ p(\alpha)$ and $p(0) \rightarrow y = 0$ hold, so by the consequence rule {true} S {y=0} holds.

To be precise we actually proved $Tr_{J_1} \vdash_{CNT}$ {true} S {y=0} where J_1 is a standard interpretation of the assertion language L.

4.4. Soundness and completeness of CNT

Before we dwell on the issue of soundness and completeness of CNT we have to specify for which assertion languages and their interpretations CNT is an appropriate proof system.

As in the previous section we assume that the assertion language L contains two sorts : data and ord. As before we have a constant 0 of type ord and a binary predicate symbol < over ord. Additionally we assume that L includes second order variables of arbitrary arity and sort. The second order variables can be bound only by the *least fixed point operator* μ provided the bound variable occurs positively in the considered formula. (Here a variable occurs *positively* in a formula if none of its occurrences in a disjunctive normal form of the formula is in the scope of a negation sign). Thus if the set variable a occurs positively in p(a) then μa.p is a well formed formula. The free variables of μa.p are those of p other than a.

An interpretation J for this type of assertion language is an ordinary two-sorted second order structure subject to the following five conditions

1. The domain $J_{\underline{data}}$ of sort data is countable (to ensure countable non-determinism,
2. The domain $J_{\underline{ord}}$ of sort ord is an initial segment of ordinals (to ensure a proper interpretation of the do-rule III),
3. The domain $J_{\underline{ord}}$ contains all countable ordinals (needed for the completeness proof),
4. The constant 0 denotes the least ordinal and the predicate symbol < denotes the strict ordering of the ordinals, restricted to $J_{\underline{ord}}$,
5. The domains of each of the set sorts contain all sets of the appropriate kind (to ensure the existence of the fixed points considered below).

The truth of the formulae of L under an interpretation J is defined in a standard way. The only nonstandard case is when a formula is of the form $\mu a.p$. We put then $\models_J \mu a.p$ iff $\models_J p[R/a]$ where R is the least fixed point of an operator naturally induced by p. Having defined the truth of the formulae of L we define the truth of the correctness formulae in the usual way.

The following theorem due to APT & PLOTKIN [3] explains why this type of assertion languages and their interpretation is of interest.

Theorem 2. *Let the assertion language* L *and its interpretation* J *satisfy the above stated conditions. Then for every correctness formula* ϕ $Tr_J \vdash_{CNT} \phi$ *iff* $\models_J \phi$.

This theorem states soundness and completeness of the proof system CNT. The soundness proof should hold for any reasonable assertion language ; it is the completeness proof which dictated the specific choice of the assertion language. The arguments used in the proofs are appropriate generalizations of those used in the soundness and completeness proofs of the system NT.

The use of ordinals in assertions requires perhaps a word of comment. It can be shown that ordinals are indeed necessary, i.e. the do-rule II is not sufficient here. For example we cannot prove the correctness formula considered in section 3.11 in a proof system in which the do-rule III is replaced by the do-rule II. In case when the assertion language L contains the language of Peano arithmetic and the domain of data values J_{data} is N, the set of natural numbers, we can exactly estimate which ordinals are needed for proofs in CNT. It turns out that exactly all *recursive ordinals* are needed. (By a recursive ordinal we mean here an ordinal attached in a natural way to a tree which can be coded by a recursive set. For equivalent characterizations see ROGERS [21].)

4.5. The issue of fairness

According to the usual semantics M_{tot} the program b:=true ; do b → skip □ b:=false od does not always terminate because the computation in which always the first guard is chosen is infinite. We can however imagine restricted forms of interpretation of programs from S_n under which the above program will always terminate.

One of such interpretations is the one under the assumption of *fairness*. In the context of programs from S_n this assumption states that in every infinite computation each guard which is infinitely often true is eventual] chosen. Here a guard is true if it evaluates to true at the moment the control in the program is just before it.

This type of assumptions is particularly important when studying the behaviour of parallel programs in the context of which fairness is a most general modeling of the fact that the ratio of speeds between concurrent processors may be arbitrarily large and varying but always finite. Study of the hypothesis of fairness in the context of nondeterministic programs is partially motivated by the fact that parallel programs can be modelled by nondeterministic programs.

We now formally define the semantics of programs from S_n under the assumption of fairness. Let $\xi = \langle S_0, \sigma_0\rangle \to \langle S_1, \sigma_1\rangle \to \ldots$ be an infinite computation starting in $\langle S_0, \sigma_0\rangle$. We say that ξ is *fair* if it fullfils the following two conditions:

i) for each program $S \equiv$ <u>if</u> $e_1 \to S_1 \square \ldots \square e_m \to S_m$ <u>fi</u> ; S' and each $i = 1,\ldots,m$ if there are infinitely many j's for which $\langle S, \sigma_j\rangle$ appears in ξ and $\models_J e_i(\sigma_j)$, then there are infinitely many j's among them such that the transition $\langle S, \sigma_j\rangle \to \langle S_i ; S', \sigma_{j+1}\rangle$ appears in ξ,

ii) for each program $S \equiv$ <u>do</u> $e_1 \to S_1 \square \ldots \square e_m \to S_m$ <u>od</u> ; S' and each $i = 1,\ldots,m$ if there are infinitely many j's for which $\langle S, \sigma_j\rangle$ appears in ξ and $\models_J e_i(\sigma_j)$, then there are infinitely many j's among them such that the transition $\langle S, \sigma_j\rangle \to \langle S_i ; S ; S', \sigma_{j+1}\rangle$ appears in ξ.

To avoid confusion resulting from the fact that various occurrences of S in ξ do not need to correspond with the same program, we should actually label each statement with a unique label. It is clear how to perform this process and we leave it to the reader.

We define the fair semantics for the programs from S_n by putting

$$\begin{aligned} M_{fair}[\![S]\!](\sigma) = {} & M[\![S]\!](\sigma) \\ & \cup \{\perp \mid \text{there exists a fair infinite computation starting in } \langle S, \sigma\rangle\} \\ & \cup \{\underline{fail} \mid S \text{ can fail from } \sigma\}. \end{aligned}$$

Thus the difference between the semantics M_{tot} and M_{fair} lies in the treatment of infinite unfair computations. We assume that all finite computations are fair.

We now define the notion of total correctness of the programs considered under the assumption of fairness by putting

$$\models_{J,fair} \{p\}\ S\ \{q\} \text{ iff } M_{fair} [\![S]\!] ([p]_J) \subseteq [q]_J$$

where of course

$$M_{fair} [\![S]\!] ([p]_J) = \bigcup_{\sigma\in[p]_J} M_{fair}[\![S]\!] (\sigma).$$

If $\models_{J,fair} \{p\}\ S\ \{q\}$ holds then we say that {p} S {q} holds under the assumption of fairness (w.r.t. J). Thus $\models_{J,fair} \{p\}\ S\ \{q\}$ holds iff each fair computation sequence of S starting in a state satisfying p successfully terminates and the terminating state satisfies q.

4.6. A transformation realizing fairness

We now wish to present a proof system in which total correctness under the assumption of fairness can be proved. For didactic resons instead of presenting the proof rules immediately, we rather explain how to derive them. To this purpose we first provide a transformation of a program $S \in \mathcal{S}_n$ into a program $S_{fair} \in \mathcal{S}_{cn}$ which realizes the assumption of fairness in the sense that S_{fair} generates exactly all fair computations of S. We proceed by the following successive steps:

1. replace each subprogram $\underline{do}\ e_1 \rightarrow S_1 \square \dots \square e_m \rightarrow S_m\ \underline{od}$ of S by

$$\underline{do} \bigvee_{i=1}^{m} e_i \rightarrow \underline{if}\ e_1 \rightarrow S_1 \square \dots \square e_m \rightarrow S_m\ \underline{fi}\ \underline{od},$$

2. replace each subprogram $\underline{if}\ e_1 \rightarrow S_1 \square \dots \square e_m \rightarrow S_m\ \underline{fi}$ of S by the following subprogram

$$\underline{for}\ j:=1\ \underline{to}\ m\ \underline{if}\ e_j\ \underline{then}\ z_j:=z_j-1\ ;$$

$$\underline{if}\ e_1 \wedge z_1 = 0 \wedge \forall_j z_j \geq 0 \rightarrow z_1:=?\ ;\ S_1$$
$$\square \dots \square e_m \wedge z_m = 0 \wedge \forall_j z_j \geq 0 \rightarrow z_m:=?\ ;\ S_m\ \underline{fi},$$

3. Rename all variables $z_1,\dots,z_m$ appropriately so that each if-construct has its "own" set of these variables.

Strictly speaking the program S_{fair} does not belong to $\mathcal{S}_{cn}$ as the if-then and the for-constructs are not assumed in the syntax. It is however clear how to change it here into a sequence of the if-constructs. Note that in step 1 we replaced each subprogram of S of the form of a do-loop by another subprogram which is equivalent to the original one *in the sense* of the M_{fair} semantics.

Let us call the subprograms introduced in step 2 the $\underline{if}_{fair}$-constructs. The above transformation boils down to building into all <u>if</u>-constructs of S a fair scheduler in which the auxiliary variables z_i count down to a moment when the corresponding guard is selected.

The following lemma relates S to S_{fair}.

<u>Lemma 2.</u>

a) *If* ξ *is a fair non failing computation of* S *then an extension* ξ' *of* ξ *dealing with the auxiliary variables of* S_{fair} *is a non failing computation of* S_{fair}.

b) *If* ξ *is a non failing computation of* S_{fair} *then its restriction to the computation steps dealing with* S *is a fair non failing computation of* S.

<u>Proof</u>

a) We annotate the states in ξ by assigning in each of them values to all variables z_i. Given a state σ_j there are two cases.

<u>Case I</u>. For no state $\sigma_k (k > j)$ the guard corresponding with z_i has been chosen.

Then by the assumption of fairness this guard has been only finitely many times enabled in case the control was there. We put $\sigma_j(z_i)$ to be equal 1 + the number of times the guard will still be enabled whenever the control will be there.

<u>Case II</u>. For some state $\sigma_k (k > j)$ the guard corresponding with z_i has been chosen. We put $\sigma_j(z_i)$ to be equal 1 + the number of times the guard will still be enabled and not chosen whenever the control will be there.

b) By the construction of S_{fair} the restriction of ξ to the computation steps dealing with S is a computation sequence for S. Suppose that this restriction is not a fair computation sequence. Then behind some point in this computation a guard would be infinitely many times enabled at the moment a control is there and yet never chosen. By the construction of S_{fair} the variable z_i corresponding with this guard would become arbitrarily small. This is however impossible because as soon as it becomes negative a failure will arise. □

<u>Corollary 2</u>. *Suppose that none of the auxiliary variables introduced in* S_{fair} *occurs free in the assertions* p *and* q. *Then*

$$\models_{J,fair} \{p\}\ S\ \{q\} \text{ iff } \forall\sigma\ [\models_J p(\sigma) \rightarrow S \text{ cannot fail from } \sigma]$$

$$\text{and } \models_{J,weak}\{p\}\ S_{fair}\ \{q\}. \quad \Box$$

Here $\models_{J,weak}\{p\}\ S_{fair}\{q\}$ holds if in the definition of the semantics $M_{tot}[\![S_{fair}]\!]$ of S we drop any mentioning of failure. We then say that S_{fair} is *weakly totally correct under* J with respect to p and q.

4.7. A proof system dealing with fairness

The above corollary indicates that in order to prove total correctness of S under the assumption of fairness it is sufficient to prove weak total correctness of S_{fair} provided the absence of failure in S can be established. To prove weak total correctness of S_{fair} we can use the proof system CNT defined in section 4.2 in which the if-rule II is replaced by the original if-rule in order to ignore the possibility of failures. Call this system CWT.

Assume now for a moment that only deterministic do-loops are allowed, i.e. do-loops of the form do e → S od. Then the first step in the transformition discussed in the previous section is not needed and can be deleted. Now, due to the form of S_{fair} any proof of its weak total correctness can be transformed into a direct proof of S provided we use the following transformed version of the if-rule:

$$\frac{\begin{array}{l}\{p\}\ \underline{\text{for}}\ j:=1\ \underline{\text{to}}\ m\ \underline{\text{if}}\ e_j\ \underline{\text{then}}\ z_j:=z_j-1\ \{p'\},\\ \{p' \wedge e_i \wedge z_i = 0 \wedge \bar{z} \geq 0\}\ z_i:=?\ ;\ S_i\{q\}\ i = 1,\dots,m\end{array}}{\{p\}\ \underline{\text{if}}\ e_1 \rightarrow S_1\ \Box \dots \Box e_m \rightarrow S_m\ \underline{\text{fi}}\ \{q\}}$$

Indeed, by applying the above rule we replace systematically each $\underline{\text{if}}_{fair}$-subprogram of S_{fair} by the original if-subprogram of S; thus, in effect we obtain a direct proof of S. The above rule can be simplified if we "absorb" all assignments to auxiliary variables into the assertion p. In such a way we obtain a proof rule dealing exclusively with the if-construct and its components.

The last issue to be dealt with is that of freedom of failure which has to be dealt with according to Corollary 2. This problem can be taken care of in the same way as in section 3.7 of by simply adding to the premises of the fair if-rule the assertion $p \rightarrow \bigvee_{i=1}^{m} e_i$.

Summerizing, the final version of the rule has the following form

RULE 11: fair if-rule

$$\frac{p \rightarrow \bigvee_{i=1}^{m} e_i, \quad \{p\,[\underline{\text{if}}\ e_j\ \underline{\text{then}}\ z_j+1\ \underline{\text{else}}\ z_j/z_j]_{j\neq i}[1/z_i] \wedge e_i \wedge \bar{z} \geq 0\}\ S_i\ \{q\}_{i=1,\ldots,m}}{\{p\}\ \underline{\text{if}}\ e_1 \rightarrow S_1 \,\square \ldots \square\, e_m \rightarrow S_m\ \underline{\text{fi}}\ \{q\}}$$

We still have to deal with the problem of do-loops as we assumed above that only deterministic loops are allowed. For this purpose we have to go back to the transformation from the previous section. In step 1 we replaced each do-loop by a program equivalent to it in the sense of the M_{fair} semantics. Therefore a proof of total correctness under the assumption of fairness of the latter program constitutes a proof of total correctness under the assumption of fairness of the former one. Thanks to this observation we can derive the fair do-rule. It has the following form after some simplifications:

RULE 12 : fair do-rule

$$\frac{\begin{array}{l} p(\alpha) \wedge \alpha > 0 \rightarrow \bigvee_{i=1}^{m} e_i, \quad p(0) \rightarrow \bigwedge_{i=1}^{m} \neg e_i, \\ \{p(\alpha)\,[\text{if}\ e_j\ \underline{\text{then}}\ z_j+1\ \underline{\text{else}}\ z_j/z_j]_{j\neq i}[1/z_i] \wedge \alpha > 0 \wedge e_i \wedge \bar{z} \geq 0\} \\ S_i \\ \{\exists \beta < \alpha\ p(\beta)\}_{i=1,\ldots,m} \end{array}}{\{\exists \alpha\ p(\alpha)\}\ \underline{\text{do}}\ e_1 \rightarrow S_1 \,\square \ldots \square\, e_m \rightarrow S_m\ \underline{\text{od}}\ \{p(0)\}}$$

The assertion $p(\alpha)$ satisfies the same condition as in rule 10.

Summarizing, the proof system FN for total correctness of programs from S_n under the assumption of fairness is obtained from the proof system N by replacing the if and do-rule by the proof rules introduced above. Note that the random assignment axiom is not needed - we used it only to derive the final form of the new rules.

The only pupose of intoducing the transformation S into S_{fair} was to derive the new rules in a straightforward way. These rules deal with the *original* programs and not their transformed versions.

4.8. Soundness and completeness of FN

The following lemma provides a proof theoretic counterpart of Corollary 2.

Lemma 3. *Suppose that none of the auxiliary variables introduced in* S_{fair} *occurs free in the assertions* p *and* q. *Then*

$$Tr_J \vdash_{FN} \{p\}\ S\ \{q\} \text{ iff } Tr_J \vdash_{CWT} \{p\}\ S_{fair}\ \{q\} \text{ and}$$

$$\forall\sigma\ [\models_J p(\sigma) \rightarrow S \text{ cannot fail from } \sigma]. \quad \Box$$

This lemma can be easily justified on the basis of remarks provided in the previous section while introducing the new proof rules.

Lemma 3 together with Corollary 2 reduces the question of soundness and completeness of FN to that of CWT. But the latter system is clearly sound and complete in the sense of section 4.4. This shows that the proof system FN is also sound and complete in the same sense. We have only to restrict additionally the class of allowed structures to those which in their data domain contain natural numbers.

4.9. An example of a proof in FN

We conclude the discussion of fairness by presenting an example proof in FN. Consider the following program S:

```
do x > 0 → if true → if b → x:=x-1
                     □ b → b:=false
                     □ ¬b → skip fi
           □ true → b:=true fi
od
```

We want to prove $\models_{J_0,fair}$ {true} S {true}, i.e. that S always terminates under the assumption of fairness.

To this purpose we have to find an assertion $p(\alpha)$ such that

(14) $p(\alpha) \wedge \alpha > 0 \rightarrow x > 0$

(15) $p(0) \rightarrow x \leq 0$

(16) $\exists \alpha\ p(\alpha)$

and

(17) $\{p(\alpha) \wedge \alpha > 0 \wedge x > 0\}\ S'\ \{\exists \beta < \alpha\ p(\beta)\}$

where S' is the body of the do-loop. (Note that we use here the original do-rule (rule 10) as the do-loop in question is deterministic. It is easy to see that the do-rules 10 and 12 are equivalent in the case of deterministic do-loops.)

Let $\rho(a,b,c,d) = \omega^3.a + \omega^2.b + \omega.c + d$ for any integers a,b,c,d where $a > 0$. Then $\rho(a,b,c,d)$ is an ordinal. We define

$$p(\alpha) \equiv \alpha = \underline{\text{if}}\ x > 0\ \underline{\text{then}}\ \rho(x, z_3, 1-b,\ b \rightarrow z_1, z_2)\ \underline{\text{else}}\ 0.$$

In the expression 1 - b, true is interpreted as 1, false as 0 ; $b \rightarrow z_1, z_2$ stands for if b then z_1 else z_2 ; the auxiliary variables z_1 and z_2 are associated with the outer guards and z_3, z_4 and z_5 with the inner guards, respectively.

It is clear that (14) - (16) hold. To prove (17) we have to insure that in a fair computation the value of ρ decreases on each iteration of the loop. More formally we wish to apply the fair if-rule so we have first to prove the premises

(18) $\{(p(\alpha) \wedge \alpha > 0 \wedge x > 0)\ [z_2+1/z_2][1/z_1] \wedge z_1, z_2 \geq 0\}\ S_1\ \{\exists \beta < \alpha\ p(\beta)\}$

and

(19) $\{p(\alpha) \wedge \alpha > 0 \wedge x > 0)\ [z_1+1/z_1][1/z_2] \wedge z_1, z_2 \geq 0\}\ b := \underline{\text{true}}\ \{\exists \beta < \alpha\ p(\beta)\}$

as the first premise of the fair <u>if</u>-rule is obviously satisfied. Here

$$S_1 \equiv \underline{\text{if}}\ b \rightarrow x:=x-1$$
$$\Box\ \ b \rightarrow b:=\underline{\text{false}}$$
$$\Box \neg b \rightarrow \underline{\text{skip}}\ \underline{\text{fi}}.$$

To prove (18) we once again wish to apply the fair <u>if</u>-rule. The premises to prove are

(20) $$\{p_1[b \rightarrow z_4+1, z_4/z_4][\neg b \rightarrow z_5+1, z_5/z_5][1/z_3] \wedge b \wedge z_3, z_4, z_5 \geq 0\}$$
$$x:=x-1\ \{\exists\, \beta \rightarrow \alpha\ p(\beta)\}$$

(21) $$\{p_1[b \rightarrow z_3+1, z_3/z_3][\neg b \rightarrow z_5+1, z_5/z_5][1/z_4] \wedge b \wedge z_3, z_4, z_5 \geq 0\}$$
$$b:=\underline{\text{false}}\ \{\exists\, \beta \rightarrow \alpha\ p(\beta)\}$$

and

(22) $$\{p_1[b \rightarrow z_i+1, z_i/z_i]_{i=3,4}[1/z_5] \wedge \neg b \wedge z_3, z_4, z_5 \geq 0\}\ \underline{\text{skip}}\ \{\exists\, \beta < \alpha\ p(\beta)\}$$

where

$$p_1 \equiv (p(\alpha) \wedge \alpha > 0 \wedge x > 0)\ [z_2+1/z_2][1/z_1] \wedge z_1, z_2 \geq 0.$$

Note that the pre-assertion of (20) is equivalent to $\rho(x,1,0,1) = \alpha \wedge b \wedge x > 0 \wedge \bar{z} \geq 0$.

We have by the assignment axiom

$$\{\rho(x,1,0,1) = \alpha \wedge b \wedge x > 0 \wedge \bar{z} \geq 0\}$$
$$x:=x-1$$
$$\{(\rho(x+1,1,0,1) = \alpha \wedge b \wedge x > 0 \wedge \bar{z} \geq 0) \vee \rho(0)\}$$

which implies by the consequence rule (20) as the necessary implication is clearly true.

To prove (21) note that the pre-assertion of (21) is equivalent to

$$\rho(x,z_3+1,0,1) = \alpha \wedge \alpha > 0 \wedge b \wedge \bar{z} \geq 0 \wedge x > 0$$

which in turn implies the assertion

$$q \equiv \exists\beta < \alpha(x > 0 \wedge \bar{z} \geq 0 \wedge \beta = \rho(x,z_3,1,z_2)).$$

Now by the assignment axiom and the consequence rule

$$\{q\}\ b{:=}\underline{\text{false}}\{\exists\beta < \alpha\ p(\beta)\}$$

so (21) by the consequence rule.

Finally, to prove (22) we note that

$$p_1[b \rightarrow z_1{+}1,\ z_i/z_i]_{i=3,4}[1/z_5] \wedge \neg b \wedge z_3,z_4,z_5 \geq 0$$

implies

$$\rho(x,z_3,1,z_2{+}1) = \alpha \wedge \neg b \wedge \bar{z} \geq 0 \wedge x > 0$$

which in turn implies $\exists\beta < \alpha\ p(\beta)$. Hence (22) holds by the skip axiom.

Now, from (20) - (22) we get (18) by the fair <u>if</u>-rule.

To prove (17) note that the pre-assertion of (19) is equivalent to

$$\rho(x,z_3,1{-}b,b \rightarrow z_1{+}1,1) = \alpha \wedge \alpha > 0 \wedge x > 0 \wedge \bar{z} \geq 0$$

which in turn implies the assertion

$$r \equiv \exists\beta < \alpha(\rho(x,z_3,0,z_1{+}1) = \beta \wedge x > 0 \wedge \bar{z} \geq 0).$$

Now by the assignment axiom and the consequence rule $\{r\}\ b{:=}\underline{\text{true}}\ \{\exists\beta < \alpha\ p(\beta)\}$ so (19) by the consequence rule.

We now proved both (18) and (19) and we get (17) by the fair <u>if</u>-rule. (14) - (17) imply by the <u>do</u>-rule {<u>true</u>} S {<u>true</u>} so by virtue of the soundness of the system FNT we get $\models_{J_0,\text{fair}}$ {<u>true</u>} S {<u>true</u>}. This concludes the proof.

<u>4.10 The issue of justice</u>

Another possible restricted interpretation of nondeterministic programs is the one under the assumption of justice. In the context of programs from S_n this assumption states that in every infinite computation each guard which is true from some moment on is eventually chosen. Here, as before, a guard is true if it evaluates to <u>true</u> at the moment the control in the program is just before it.

The assumption of *justice* can be treated in an analogous way as that of fairness. To obtain a transformation realizing justice we only need to replace in the transformation from section 4.6. the program from the first line in step 2 by

$$\underline{\text{for}}\ j:=1\ \underline{\text{to}}\ m\ \underline{\text{if}}\ e_j \rightarrow z_j:=z_j-1 \ \square\ \neg e_j \rightarrow z_j:=?\ \underline{\text{fi}}.$$

All other steps in the development of the proof rules for justice are the same as before and left to the reader.

As a final remark we would like to indicate that in the transformation from section 4.6 we can omit the conditions $z_i=0$ from all of the guards, both for the case of fairness and justice. Clearly various other transformation also satisfy lemma 2. We chose here a transformation which leads to simplest proof rules dealing with fairness or justice.

5. BIBLIOGRAPHICAL REMARKS

The first treatment of nondeterminism in the framework of Hoare's logic is due to LAUER [15] where a proof rule dealing with the or-construct (the meaning of the construct S_1 or S_2 is execute either S_1 or S_2) is introduced. Correctness of nondeterministic programs introduced in section 3 is extensively studied in DIJKSTRA [8] using a different approach. Axioms 1,2 and proof rules 3,6 are from HOARE [14]. Rules 4,5 are obvious modifications of the appropriate rules dealing with the deterministic versions of the constructs and introduced in LAUER [15] and HOARE [14], respectively. They appear for example in DE BAKKER [5] (p. 292).

Soundness and completeness proofs from sections 3.5 and 3.6 are straightforward generalizations of the corresponding proofs dealing with deterministic versions of the programs and presented for example in DE BAKKER [5] (section 3). Rule 7 is inspired by the discussion of clean behaviour of programs in PNUELI [19]. The completeness proof from section 3.8 is an appropriate modification of a corresponding proof from HAREL [11].

The notion of bounded nondeterminism is introduced in DIJKSTRA [8]. Countable nondeterminism is extensively studied in APT & PLOTKIN [3] and several related references can be found there. Corollary 1 is implicit in DIJKSTRA [8]. Axiom 9 is from HAREL [11] and rule 10 from APT & PLOTKIN [3] where a slightly different syntax is used. Sections 4.3 and 4.4 are based on APT & PLOTKIN [3], as well. The program from section 4.3 is from DIJKSTRA [8].

The issue of fairness is discussed in several papers (see for example PNUELI [19]). First proof rules dealing with fairness were proposed in GRÜMBERG et al. [10], LEHMANN et al. [17] and APT & OLDEROG [2]. LEHMANN [16] contains a simplified completeness proof of a rule introduced in GRÜMBERG et al. [10]. Sections 4.7 - 4.10 are based on APT et al. [4]. Transformations realizing fairness were first introduced in APT & OLDEROG [2]. Simplified versions of such transformations are given and discussed in PARK [18].

The program studied in section 4.9 is due to S. Katz. First proof rules dealing with justice were proposed in APT & OLDEROG [3] and LEHMANN et al. [17]. LEHMANN [16] contains another proof rule for justice. In LEHMANN et al. [17] arguments for introducing the hypotheses of justice and fairness when studying parallel programs are given. QUEILLE & SIFAKIS [20] contains a thorough discussion of various possible formalizations of the assumption of fairness.

REFERENCES

[1] APT, K.R., *Ten Years of Hoare's Logic, a survey, part I*, TOPLAS, vol. 3, N° 4, pp. 431-483 (1981).

[2] APT, K.R. & E.-R. OLDEROG, *Proof rules dealing with fairness*, Bericht Nr. 8104, Inst. Inf. Prakt. Math., University of Kiel, (1981). (Extended abstract appeared in: Logic of Programs, Lecture Notes in Computer Science, 131, pp. 1-8, Springer-Verlag, New York, (1982).)

[3] APT, K.R. & G.D. PLOTKIN, *Countable nondeterminism and random assignment*, Technical Report 82-7, LITP, Université Paris 7 (1982). (Extended abstract appeared as : A Cook's tour of countable nondeterminism, in : Proceedings ICALP'81, Lecture Notes in Computer Science, 115, pp. 479-494, Springer Verlag, Berlin, (1981).)

[4] APT, K.R., A. PNUELI & J. STAVI, *Fair termination revisited - with delay*, in: Proc. 2nd Conference FST and TCS, Bangalore, India, pp. 146-170 (1982).

[5] DE BAKKER, J.W., *Mathematical theory of program correctness*, Prentice-Hall, Englewood Cliffs, (1980).

[6] COOK, S.A., *Soundness and completeness of an axiom system for program verification*, SIAM J. Comput., vol. 7, n° 1, pp. 70-90 (1978).

[7] DIJKSTRA, E.W., *Guarded commands, nondeterminacy and formal derivation of programs*, Communications ACM, vol. 18, N° 8 (1975).

[8] DIJKSTRA, E.W., *A discipline of programming*, Prentice-Hall Englewood Cliffs, (1976).

[9] FLOYD, R.W., *Nondeterministic algorithms*, Journal ACM, vol. 14, N° 4, pp. 636-644 (1967).

[10] GRÜMBERG, O., N. FRANCEZ, J.A. MAKOWSKY & W.P. de ROEVER, *A proof rule for fair termination of guarded commands, in : Algorithmic languages* (Eds, J.W. de BAKKER, J.C. van VLIET), pp. 399-416, IFIP, North Holland, Amsterdam (1981).

[11] HAREL, D., *First-order dynamic logic*, Lecture Notes in Computer Science, 68, Springer Verlag, New York (1979).

[12] HAREL, D. & D. KOZEN, *A programming language for the inductive sets, and applications*, in : Proceedings ICALP'82, Lecture Notes in Computer Science, 140, Springer-Verlag, Berlin

[13] HENNESSY, M.C.B. & G.D. PLOTKIN, *Full abstraction for a simple programming language*, in : Proc. 8th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, 74, pp. 108-120, Springer Verlag, New-York (1979).

[14] HOARE, C.A.R., *An axiomatic basis for computer programming*, Communications ACM, vol. 12, N° 10, pp. 576-580, 583 (1969).

[15] LAUER, P.E., *Consistent formal theories of the semantics of programming languages*, Technical Report TR. 25.121, IBM Lab. Vienna, (1971).

[16] LEHMANN, D., *Another proof for the completeness of a rule for the fair termination of guarded commands and another rule for their just termination*, Technical Report IW 178/81, Mathematisch Centrum, (1981).

[17] LEHMANN, D., PNUELI, A. & J. STAVI, *Impartiality, justice and fairness: the ethics of concurrent termination*, Proceedings ICALP'81, Lecture Notes in Computer Science, 115, pp. 264-277, Springer-Verlag, Berlin, (1981).

[18] PARK, D., *A predicate transformer for weak fair iteration, in :* Proceedings 6^{th} IBM Symposium on Mathematical Foundations of Computer Science, Hakone, Japan (1981).

[19] PNUELI, A., *The temporal semantics of concurrent programs,* Theoretical Computer Science, vol. 13, N^o 1, pp. 45-60, (1981).

[20] QUEILLE, J.P. & J. SIFAKIS, *Fairness and related properties in transition systems - a time logic to deal with fairness,* Technical Report - RR N^o 292, University of Grenoble (1982).

[21] ROGERS, H. Jr, *Theory of recursive functions and effective computability,* McGraw-Hill, New York, (1967).