Theoretical Computer Science 28 (1984) 83–109
North-Holland

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

Krzysztof R. APT

L.I.T.P., Université Paris 7, 2, Place Jussieu, 75251 Paris, France

Communicated by M. Nivat
Received September 1982
Revised February 1983

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 Part I by Apt (1981), where various results concerning Hoare's approach to proving correctness of sequential programs are presented.

Key words. Hoare's logic, partial correctness, total correctness, soundness, completeness, bounded nondeterminism, countable nondeterminism, freedom of failure, fairness.

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 [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 [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 [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 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 [1, Section 2]. 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. In Section 5 we attempt to assess Hoare's approach to sequential and nondeterministic programs. Finally, in Section 6, bibliographical remarks are provided.

2. Preliminaries

Throughout this 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 p, q, r. Simple variables are denoted by a, b, x, y, z, expressions by 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 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 $\perp$ reporting nontermination of a program and **fail** reporting a failure in execution of a program. We have, by definition, $\not\models_J p(\perp)$, $\not\models_J p(\mathbf{fail})$ for all formulae p. We define $[p]_J$ to be the set of 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, $\perp \notin [p]_J$ and $\mathbf{fail} \notin [p]_J$.

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

3. Bounded nondeterminism

Denote by $\mathscr{S}_n$ the least class of programs such that, for all Boolean expressions $e_1, \ldots, e_m$ and $S_1, \ldots, S_m \in \mathscr{S}_n$,

$$\mathbf{if}\ e_1 \to S_1 \square \cdots \square e_m \to S_n\ \mathbf{fi} \in \mathscr{S}_n$$

and

$$\textbf{do } e_1 \to S_1 \square \cdots \square e_m \to S_m \textbf{ od} \in S_n.$$

This class of programs was introduced in [7] and further extensively studied in [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** $e_1 \to S_1 \square \cdots \square e_m \to S_m$ **fi** is the following: 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** $e_1 \to S_1 \square \cdots \square e_m \to S_m$ **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** $e_1 \to S_1 \square \cdots \square e_m \to S_m$ **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 $\mathscr{S}_n$ we define their semantics. We follow here the approach of Hennessy and 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 '$\to$' between pairs $\langle S, \sigma\rangle$ consisting of a program S and a state σ. The intuitive meaning of the relation

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

is the following: 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 the considered step of S_1 leads to state τ with S_1 properly or improperly terminated. We assume that, for any S, $E; S = S; E = S$.

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

(i) $\langle \textbf{skip}, \sigma\rangle \to \langle E, \sigma\rangle$,

(ii) $\langle x := t, \sigma\rangle \to \langle E, \tau\rangle$ where $\tau(x) = \sigma(t)$ and $\tau(y) = \sigma(y)$ for $y \not\equiv x$,

(iii) $\langle \textbf{if } e_1 \to S_1 \square \cdots \square e_m \to S_m \textbf{ fi}, \sigma\rangle \to \langle S_i, \sigma\rangle$ if $\models_I e_i(\sigma)$,

(iv) $\langle \textbf{if } e_1 \to S_1 \square \cdots \square e_m \to S_m \textbf{ fi}, \sigma\rangle \to \langle E, \textbf{fail}\rangle$ if $\models_I \bigwedge_{i=1}^{m} \neg e_i(\sigma)$,

(v) $\langle \textbf{do } e_1 \to S_1 \square \cdots \square e_m \to S_m \textbf{ od}, \sigma\rangle \to \langle S_i; \textbf{do } e_1 \to S_1 \square \cdots \square e_m \to S_m \textbf{ od}, \sigma\rangle$ if $\models_I e_i(\sigma)$,

(vi) $\langle \textbf{do } e_1 \to S_1 \square \cdots \square e_m \to S_m \textbf{ od}, \sigma\rangle \to \langle E, \sigma\rangle$ *if* $\models_I \bigwedge_{i=1}^{m} \neg e_i(\sigma)$,

(vii) if $\langle S_1, \sigma\rangle \to \langle S_2, \tau\rangle$ then $\langle S_1; S, \sigma\rangle \to \langle S_2; S, \tau\rangle$.

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

We now introduce the following definitions.

Definition 3.1. (i) S can *diverge* from σ if there exists an infinite sequence $\langle S_i, \sigma_i\rangle$ $(i = 0, 1, \ldots)$ such that $\langle S, \sigma\rangle = \langle S_0, \sigma_0\rangle \to \langle S_1, \sigma_1\rangle \to \cdots$. Such a sequence is called an *infinite computation starting in* $\langle S, \sigma\rangle$.

(ii) S can *fail* from σ if

$$\langle S, \sigma\rangle \to^* \langle S_1, \mathbf{fail}\rangle \quad \text{for some } S_1.$$

(iii) A finite sequence $\langle S_i, \sigma_i\rangle$ $(i = 0, 1, \ldots, k)$ such that $\langle S, \sigma\rangle = \langle S_0, \sigma_0\rangle \to \langle S_1, \sigma_1\rangle \to \cdots \to \langle S_k, \sigma_k\rangle$ and $S_k \equiv E$ or $\sigma_k = \mathbf{fail}$ is called a ***computation starting in*** $\langle S, \sigma\rangle$ of *length* k. If $\sigma_u \neq \mathbf{fail}$, then it is a ***non failing computation.***

The following lemma will be needed later.

Lemma 3.2. *If S cannot diverge from σ, then there exists a natural number k such that all computations starting in $\langle S, \sigma\rangle$ are of length at most k.*

Proof. Consider the set of all finite sequences $\langle S, \sigma\rangle = \langle S_0, \sigma\rangle_0 \to \cdots \to \langle S_n, \sigma_n\rangle$ 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. □

We now define three types of semantics for the programs from $\mathcal{S}_n$ by putting

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

$$\mathcal{M}_{\mathrm{wtot}}[\![S]\!](\sigma) = \mathcal{M}[\![S]\!](\sigma) \cup \{\perp \mid S \text{ can diverge from } \sigma\}$$

and

$$\mathcal{M}_{\mathrm{tot}}[\![S]\!](\sigma) = \mathcal{M}_{\mathrm{wtot}}[\![S]\!](\sigma) \cup \{\mathbf{fail} \mid S \text{ can fail from } \sigma\}.$$

All semantics depend on the interpretation J but we do not mention this dependence hoping that no confusion will arise. The difference between them 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 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

$$\mathcal{M}_h[\![S]\!]([\![p]\!]_I) = \bigcup_{\sigma \in [\![p]\!]_I} \mathcal{M}_h[\![S]\!](\sigma) \quad \text{for } h \in \{\mathrm{wtot}, \mathrm{tot}\} \text{ or } h \text{ empty.}$$

We then define

$$\models_{J,h}\{p\}S\{q\} \quad \text{iff} \quad \mathcal{M}_h[\![S]\!]([p]_J)\subseteq[q]_J.$$

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; $\models_{J,\mathrm{wtot}}\{p\}S\{q\}$ in addition guarantees that any execution of S starting in a state satisfying p terminates and $\models_{J,\mathrm{tot}}\{p\}S\{q\}$ guarantees that in addition no failure arise. If $\models_J\{p\}S\{q\}$ holds, we say that the program S is *partially correct under* J; if $\models_{J,\mathrm{wtot}}\{p\}S\{q\}$ holds, we say that the program S is *weakly totally correct under* J and if $\models_{J,\mathrm{tot}}\{p\}S\{q\}$ holds, we say that the program S is *totally correct under* J (all with respect to p and q).

The notion of weak total correctness is rarely used and we shall not discuss it extensively. The reasons for introducing it here will become clear in Section 4.

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

$$\{p\}\,\mathbf{skip}\,\{p\}.$$

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

$$\frac{\{p\wedge e_i\}S_i\{q\},\ i=1,\ldots,m}{\{p\}\,\mathbf{if}\ e_1\to S_1\ \cdots\ e_m\to S_m\ \mathbf{fi}\{q\}}$$

RULE 5: **do**-rule

$$\frac{\{p\wedge e_i\}S_i\{p\},\ i=1,\ldots,m}{\{p\}\,\mathbf{do}\ e_1\to S_1\ \cdots\ e_m\to S_m\ \mathbf{od}\left\{p\wedge\bigwedge_{i=1}^{m}\neg e_i\right\}}$$

p is called a *loop invariant.*

RULE 6: consequence rule

$$\frac{p\to p_1,\{p_1\}S\{q_1\},q_1\to q}{\{p\}S\{q\}}.$$

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 $2|x \vee 3|x \rightarrow$
  if $2|x \rightarrow x := x/2; a := a+1$
  $\square 3|x \rightarrow x := x/3; b := b+1$
  $\square 4|x \rightarrow x := x/4; a := a+2$ **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

$$\mathrm{Tr}_{J_0} \vdash_N \{a = 0 \wedge b = 0 \wedge x = z\} S \{z = x \cdot 2^a \cdot 3^b \wedge \neg(2|x \vee 3|x)\} \tag{1}$$

where J_0 is the standard interpretation of the language of Peano arithmetic augmented with the division operator 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

$$a = 0 \wedge b = 0 \wedge x = z \rightarrow p, \tag{2}$$

$$\{p \wedge (2|x \vee 3|x)\} S_1 \{p\} \quad \text{where } S_1 \text{ is the loop body}, \tag{3}$$

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

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 (1) by the consequence. Both (2) and (4) are obvious.

To show (3) we have to show

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

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

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

and apply the **if**-rule.

We now prove (5). By the assignment axiom

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

and

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

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 operator does not result in producing non-integer values, we should actually use here the following assignment rule in the case of division operation:

$$\frac{p[(a/b)/x] \rightarrow b \mid a}{\{p[(a/b)/x]\} x := a/b \{p\}}$$

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 3.3. *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 $\mathrm{Tr}_J \vdash_N \phi$, then $\models_J \phi$.

We call a correctness formula *valid* if it is true under all interpretations J and a proof rule *sound* if, for all interpretations J, the truth under J of its premises implies the truth under J of its conclusion.

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 then follows 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 **do** $e_1 \rightarrow S_1 \; \cdots \; e_m \rightarrow S_m$ **od**. Fix an interpretation J and assume that all the premises of the **do**-rule are true under J, i.e., that

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

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

$$\langle S, \sigma_0 \rangle \rightarrow^* \langle S, \sigma_1 \rangle \rightarrow^* \cdots \rightarrow^* \langle S, \sigma_l \rangle \rightarrow \langle E, \sigma_l \rangle$$

where $\sigma = \sigma_0$, $\tau = \sigma_l$ and, for all $j = 0, \ldots, l-1$, $\sigma_j \in [e_{k_j}]_J$ and $\sigma_{j+1} \in \mathcal{M}[\![S_{k_j}]\!](\sigma_j)$ for some $k_j \in \{1, \ldots, m\}$ and $\sigma_l \in [\bigwedge_{i=1}^m \neg e_i]_J$. We have $\sigma_0 \in [p]_J$ and if, for some $j \in \{0, \ldots, l-1\}$, $\sigma_j \in [p]_J$, then, by (8), $\sigma_{j+1} \in \mathcal{M}[\![S_{k_j}]\!]([p \wedge e_{k_j}]_J] \subseteq [p]_J$, i.e., $\sigma_{j+1} \in [p]_J$.

Thus, for all $j=0,\ldots,l$, $\sigma_j \in [p]_J$. In particular $\sigma_l \in [p]_J$ 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 Theorem 3.3 can be proved only for a special type of interpretations J. This issue is discussed at length in [1, 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):

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

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 [1]).

Definition 3.4. A proof system G for $\mathcal{S}_0$ 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 $\mathcal{S}_0$ is complete in the sense of Cook. The proof proceeds by the 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 hypotheses 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** $e_1 \to S_1 \cdots e_m \to S_m$ **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 any $i = 1, \ldots, m$ as, for *all* σ,

$$\mathcal{M}[\![\textbf{if } e_i \to S_i \square \neg e_i \to \textbf{skip fi}; S]\!](\sigma) \subseteq \mathcal{M}[\![S]\!](\sigma)$$

clearly holds. Now, since r defines $\mathrm{pre}_J(S, q)$, then, as in the case treated above, $\models_J \{r\}$ **if** $e_i \to S_i \square \neg e_i \to$ **skip 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 I. This concludes the proof.

3.7. A proof system for total correctness

To prove total correctness of programs from $\mathcal{S}_n$ we must provide proof rules ruling out possibility of failure and nontermination.

A possible failure in an execution of a program from $\mathcal{S}_n$ 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 absence of failure. We only need to ensure that at each moment when 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

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

A possible nontermination of an execution of a program from $\mathcal{S}_n$ 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 [11] where a different formalism is used.

RULE 8: **do**-rule II

$$p(n) \wedge n > 0 \to \bigvee_{i=1}^{m} e_i,\ p(0) \to \bigwedge_{i=1}^{m} \neg e_i,$$

$$\frac{\{p(n) \wedge n > 0 \wedge e_i\} S_i \{\exists m < n\, p(m)\}_{i=1,\ldots,m}}{\{\exists n\, p(n)\} \textbf{do } e_1 \to S_1 \square \cdots \square e_m \to S_m \textbf{ od} \{p(0)\}}.$$

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 $\mathcal{S}_n$.

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

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

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

We now have

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

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

$$p(0) \rightarrow \neg(2|x \vee 3|x), \tag{12}$$

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

where the last correctness formula can be proved using the **if**-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 **do**-rule II, (10) and (12) we obtain (1) as desired.

3.8. Arithmetical soundness and completeness of NT

As explained in [1, Section 2.11] 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 [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 p_0 of L^+ which, when interpreted under J, provides the ability to encode finite sequences of elements from the domain of J into one element. (The last requirement is needed only for the completeness proof.)

More formally, p_0 satisfies the following condition for any natural number n,

$$\models_J \forall x_1, \ldots, x_m\, \exists y\, \forall x\, \forall i\, (\mathrm{nat}(i) \wedge i \leq n \rightarrow (p_0(x, i, y) \leftrightarrow x = x_i))$$

where x, i, y are the free variables of p_0.

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 $\mathrm{nat}(n) \wedge p(n)$.

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

It can be shown that the proof system NT is arithmetically sound. The case of the **if**-rule II is easily handled. The proof of soundness of the **do**-rule II for the case of arithmetical interpretations is in turn an easy modification of the proof of

soundness of the **do**-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, $\models_{J,\mathrm{tot}} \phi$ implies $\mathrm{Tr}_J \vdash_G \phi$.

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

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

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

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 $\mathrm{pret}_J(S, q)$. This fact is not completely obvious as the definition of $\mathrm{pret}_J(S, q)$ also mentions (the nonexistence of) infinite sequences. This difficulty, however, can be resolved by making use of Lemma 3.2 thanks to which we arrive at the following alternative definition of $\mathrm{pret}_J(S, q)$ amenable to be coded:

$$\begin{aligned}\mathrm{pret}_J(S, q) = \{\sigma: \forall \tau [(\langle S, \sigma\rangle \to^* \langle E, \tau\rangle) \to \models_J q(\tau)] \\ \text{and } \exists k \neg \exists S_0, \ldots, S_{k+1}, \sigma_0, \ldots, \sigma_{k+1} \\ [S = S_0, \sigma = \sigma_0 \wedge \forall i \leq k (\langle S_i, \sigma_i\rangle \to \langle S_{i+1}, \sigma_{i+1}\rangle)] \\ \text{and } \neg \exists S' (\langle S, \sigma\rangle \to^* \langle S', \mathbf{fail}\rangle)\}.\end{aligned}$$

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 **if**- and **do**-constructs. Let J be an arithmetical interpretation.

If $\models_{J,\mathrm{tot}} \{p\}\, \mathbf{if}\, e_1 \to S_1 \,\square\, \cdots \,\square\, e_m \to S_m\, \mathbf{fi}\, \{q\}$, then by definition $\models_J p \to \bigvee_{i=1}^m e_i$ and $\models_{J,\mathrm{tot}} \{p \wedge e_i\} S_i \{q\}$ for $i = 1, \ldots, m$. By the induction hypothesis $\mathrm{Tr}_J \vdash_{\mathrm{NT}} \{p \wedge e_i\} S_i \{q\}$ for $i = 1, \ldots, m$ so, by **if**-rule II, $\mathrm{Tr}_J \vdash_{\mathrm{NT}} \{p\}\, \mathbf{if}\, e_1 \to S_1 \,\square\, \cdots \,\square\, e_m \to S_m\, \mathbf{fi}\, \{q\}$.

Assume now $\models_{J,\mathrm{tot}} \{r\} S \{q\}$ where $S \equiv \mathbf{do}\, e_1 \to S_1 \,\square\, \cdots \,\square\, e_m \to S_m\, \mathbf{od}$. Let n be a fresh variable. Let now C be the following set of states:

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

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 definition of $p(n)$ we now have $\models_J p(n) \wedge n > 0 \to \bigvee_{i=1}^m e_i$, $\models_J p(0) \to \bigwedge_{i=1}^m \neg e_i$. 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 induction hypothesis and **do**-rule II we get

$$\mathrm{Tr}_J \vdash_{\mathrm{NT}} \{\exists n\, p(n)\}\, \mathbf{do}\, e_1 \to S_1 \,\square\, \cdots \,\square\, e_m \to S_m\, \mathbf{od}\, \{p(0)\}.$$

We now have by assumption $[r]_J \subseteq \text{pret}_J(S, q)$ and so, by virtue of Lemma 3.2, $\models_J r \to \exists n p(n)$. Also $\models_J p(0) \to q$ holds so by the consequence rule we get $\text{Tr}_J \vdash_{\text{NT}} \{r\}$ **do** $e_1 \to S_1 \square \cdots \square e_m \to S_m$ **od** $\{q\}$.

This concludes the proof.

3.9. Weak total correctness

In the above analysis we omitted the issue of weak total correctness. An appropriate proof system to study this notion is clearly a weaker version of NT in which the original **if**-rule is retained. We call this system WT. This system is clearly arithmetically sound and complete.

4. Countable nondeterminism

4.1. Bounded nondeterminism versus finite and countable nondeterminism

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 $\{\langle S_1, \sigma_1\rangle : \langle S, \sigma\rangle \to \langle S_1, \sigma_1\rangle\}$ 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 [9]. (Of course, we assume here that the programs are interpreted under a standard interpretation in natural numbers).

It should be noted, however, 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

$$b := \textbf{true};\ x := 0;\ \textbf{do}\ b \wedge x < y \to x := x + 1 \ \square\ b \wedge x \leq y \to b := \textbf{false}\ \textbf{od}.$$

Consequently the study of finite nondeterminism (in the above sense) can be reduced to the study of bounded nondeterminism.

This is not the case any more 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 assignment* $x := ?$ which sets to x an arbitrary nonnegative integer.

It is obvious how to define the semantics $\mathcal{M}_{\text{tot}}[\![x := ?]\!]$ of $x := ?$. We have $\perp \notin \mathcal{M}_{\text{tot}}[\![x := ?]\!](\sigma)$ for any σ. We now claim that there is no program $S \in \mathcal{S}_n$ such that $\mathcal{M}_{\text{tot}}[\![x := ?]\!] = \mathcal{M}_{\text{tot}}[\![S]\!]$. This immediately follows from the following corollary to Lemma 3.2.

Corollary 4.1. *For any $S \in \mathscr{S}_n$ and σ if $\perp \notin \mathscr{M}_{tot}[\![S]\!](\sigma)$, then $\mathscr{M}_{tot}[\![S]\!](\sigma)$ 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 we have clearly

$$\mathscr{M}[\![x := ?]\!] = \mathscr{M}[\![b := \textbf{true}; x := 0; \textbf{do } b \to x := x + 1 \square b \to b := \textbf{false od}]\!].$$

(Here and elsewhere we ignore the fact that the values of the auxiliary variables (here b) have 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., [12]).

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

Consider now the class $\mathscr{S}_{cn}$ of programs which differs from $\mathscr{S}_n$ in that additionally the instruction $x := ?$ is allowed. We now present a proof system which allows us to prove total correctness of programs from $\mathscr{S}_{cn}$. 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 **do**-rule II by its following generalization:

RULE 10: **do**-rule III

$$p(\alpha) \wedge \alpha \geq 0 \to \bigvee_{i=1}^{m} e_i,\ p(0) \to \bigwedge_{i=1}^{m} \neg e_i,$$

$$\frac{\{p(\alpha) \wedge \alpha \geq 0 \wedge e_i\}\, S_i\, \{\exists \beta < \alpha\, p(\beta)\},\ i = 1, \ldots, m}{\{\exists \alpha\, p(\alpha)\}\ \textbf{do } e_1 \to S_1 \square \cdots \square e_m \to S_m\ \textbf{od}\ \{p(0)\}}$$

where $p(\alpha)$ 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 the following program:

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

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

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 $\bar{}$ of sort (**data**, **ord**) converting integers into ordinals and a constant ω of sort **ord**. We have $\forall x(x < \omega)$ 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 bigger or equal to the number of possible iterations performed by the loop when starting in σ.

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

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

$$\rightarrow \alpha = \bar{y} \wedge y > 0 \wedge \exists z (z = y - 1 \wedge p(\bar{z})[y - 1/y])$$

$$\rightarrow \exists \beta < \alpha p(\beta)[y - 1/y].$$

By **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, $\{\mathbf{true}\}\, S \{y = 0\}$ holds.

To be precise we actually proved $\mathrm{Tr}_{J_1} \vdash_{\mathrm{CNT}} \{\mathbf{true}\}\, S \{y = 0\}$ 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. If the set variable a occurs positively in $p(a, u_1, \ldots, u_n)$ and $a(u_1, \ldots, u_n)$ is a well formed formula, then $\mu a(u_1, \ldots, u_n).p$ is a well formed formula. The free variables of $\mu a(u_1, \ldots, u_n).p$ 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 conditions.

(1) The domain $J_{\mathbf{data}}$ of sort **data** is countable (to esure countable nondeterminism).

(2) The domain $J_{\mathbf{ord}}$ of sort **ord** is an initial segment of ordinals (to ensure a proper interpretation of **do**-rule III).

(3) The domain $J_{\mathbf{ord}}$ 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 $J_{\mathbf{ord}}$.

(5) The set domain contains all sets of the appropriate kinds (to ensure the existence of the fixed points considered below).

The truth of the formulae of L under interpretation J is defined in a standard way. The only nonstandard case is when a formula is of the form $\mu a.p$. We then put $\models_J \mu a(u_1, \ldots u_n).p$ iff $\models_J p[R/a]$ where R is the least fixed point of the operator $\{p\}$ naturally induced by p:

$$\{p\}(Q) = \{\langle i_1, \ldots, i_n\rangle | \models_J p[Q/a][i_1/u_1, \ldots, i_n/u_n]\}.$$

Having defined the truth of the formulae of L we define the truth of the correctness formulae in the usual way.

The following theorem proved in [3] explains why this type of assertion languages and their interpretations is of interest.

Theorem 4.2. *Let the assertion language L and its interpretation J satisfy the above stated conditions. Then, for every correctness formula ϕ, $\mathrm{Tr}_J \vdash_{\mathrm{CNT}} \phi$ iff $\models_J \phi$.*

This theorem states soundness and completeness of the proof system CNT.

The soundness proof is a simple generalization of the corresponding proof dealing with system CNT. The completeness proof as usual proceeds by induction and only the case of the **do**-loops requires an explanation. Suppose $\models_J \{r\}\mathbf{do}\, e_1 \to S_1 \cdots e_m \to S_m\, \mathbf{od}\{q\}$.

The computations of the program S starting in a state σ form an infinitely branching tree. If S cannot diverge from σ, then this tree is well founded. With each such tree we can naturally associate a (possibility infinite) ordinal.

Similarly as in the completeness proof of the system NT consider the following set of states:

$$\mathrm{pret}_J(S, q) \cap \{\sigma\colon S \text{ cannot diverge from } \sigma \text{ and the corresponding ordinal is } \alpha\}$$

One can show that there exists an assertion $p(\alpha)$ which defines this set of states within each interpretation here considered.

It is easy to see that $p(\alpha)$ satisfies the premises of Rule 10. By induction hypothesis these premises are provable in CNT and hence the conclusion of the rule, as well. Similarly as in Section 3.8 both $\models_J r \to \exists \alpha\, p(\alpha)$ and $\models_J p(0) \to q$. By the consequence rule we now get

$$\mathrm{Tr}_J \vdash_{\mathrm{CNT}} \{r\}\mathbf{do}\, e_1 \to S_1 \cdots e_m \to S_m\, \mathbf{od}\{q\}$$

as desired.

The use of ordinals in assertions perhaps requires a word of comment. It can be shown that ordinals are indeed necessary, i.e., **do**-rule II is not sufficient here. For example we cannot prove the correctness formula considered in Section 4.3 in a proof system in which **do**-rule III is replaced by **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 to a tree which can be coded by a recursive set. For equivalent characterizations, see [21].)

4.5. Weak total correctness of countably nondeterministic programs

We conclude this discussion of countable nondeterminism by mentioning the notion of weak total correctness of programs from $\mathscr{S}_{cn}$. This notion is defined analogously as in Section 3.2.

Let CWT stand for a proof system which differs from CNT in that the original **if**-rule (RULE 4) is used in it instead of RULE 7. Clearly this proof system is sound and complete in the above sense with respect to the $\mathscr{M}_{wtot}$ semantics. This system will be useful when dealing with the issue of fairness.

4.6. The issue of fairness

According to the usual semantics $\mathscr{M}_{tot}$ the program $b := \mathbf{true}; \mathbf{do}\ b \to \mathbf{skip} \square b := \mathbf{false}\ \mathbf{od}$ does not always terminate because the computation in which the first guard is always chosen is infinite. However, we can imagine restricted forms of interpretation of programs from $\mathscr{S}_n$ 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 $\mathscr{S}_n$ this assumption states that in every infinite computation each guard which is infinitely often true is eventually 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 $\mathscr{S}_n$ under the assumption of fairness. Let $\xi = \langle S_0, \sigma_0\rangle \to \langle S_1, \sigma_1\rangle \to \cdots$ be an infinite computation starting in $\langle S_0, \sigma_0\rangle$. We say that ξ is *fair* if it fulfills the following conditions:

(i) for each program $S \equiv \mathbf{if}\ e_1 \to S_1 \square \cdots \square e_m \to S_m\ \mathbf{fi}; S'$ 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 $S \equiv \mathbf{do}\ e_1 \to S_1 \square \cdots \square e_m \to S_m\ \mathbf{od}; S'$ 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.

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 two fair semantics for the programs from $\mathscr{S}_n$ by putting

$$\mathscr{M}_{\mathrm{wfair}}[\![S]\!](\sigma) =$$

$$= \mathscr{M}[\![S]\!](\sigma) \cup \{\perp \mid \text{there exists a fair infinite computation starting in } \langle S, \sigma\rangle\},$$

$$\mathscr{M}_{\mathrm{fair}}[\![S]\!](\sigma) = \mathscr{M}_{\mathrm{wfair}}[\![S]\!](\sigma) \cup \{\mathbf{fail} \mid S \text{ can fail from } \sigma\}.$$

Thus the difference between the semantics $\mathscr{M}_{\mathrm{wtot}}$, $\mathscr{M}_{\mathrm{tot}}$ and $\mathscr{M}_{\mathrm{wfair}}$, $\mathscr{M}_{\mathrm{fair}}$ respectively lies in the treament of infinite unfair computations. We assume that all finite computations are fair.

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

$$\models_{J,h}\{p\}S\{q\} \text{ iff } \mathscr{M}_h[\![S]\!]([p]_J) \subseteq [q]_J \quad \text{for } h \in \{\mathrm{wfair}, \mathrm{fair}\}$$

where of course $\mathscr{M}_h[\![S]\!]([p]_J) = \bigcup_{\sigma \in [p]_J} \mathscr{M}_h[\![S]\!](\sigma)$.

If $\models_{J,\mathrm{fair}}\{p\}S\{q\}$ ($\models_{J,\mathrm{wfair}}\{p\}S\{q\}$) holds, then we say that S is totally (weakly totally) correct under the assumption of fairness with respect to p and q. Thus $\models_{J,\mathrm{fair}}\{p\}S\{q\}$ holds iff each fair computation of S starting in a state satisfying p successfully terminates and the terminating state satisfies q.

4.7. *A transformation enforcing fairness*

We now wish to present a proof system in which total correctness under the assumption of fairness can be proved. For didactic reasons 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 $S \in \mathscr{S}_n$ into a program $S_{\mathrm{fair}} \in \mathscr{S}_{n+1}$ which realizes the assumption of fairness in the sense that non-failing computations of S_{fair} coincide exactly with fair and non-failing computations of S. We proceed by the following successive steps:

Step 1. Replace each subprogram $\mathbf{do}\ e_1 \to S_1 \ \cdots \ e_m \to S_m\ \mathbf{od}$ of S by

$$\mathbf{do} \bigvee_{i=1}^{m} e_i \to \mathbf{if}\ e_1 \to S_1 \ \cdots \ e_m \to S_m\ \mathbf{fi}\ \mathbf{od}.$$

Step 2. Replace each subprogram $\mathbf{if}\ e_1 \to S_1 \ \cdots \ e_m \to S_m\ \mathbf{fi}$ of S by the following subprogram:

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

$$\mathbf{if}\ e_1 \wedge z_1 = 0 \wedge \forall_j z_j > 0 \to z_1 := ?; S_1 \ \cdots$$

$$e_m \wedge z_m = 0 \wedge \forall_j z_j > 0 \to z_m := ?; S_m\ \mathbf{fi}.$$

Step 3. Rename all variables $z_1, \ldots, z_m$ appropriately so that each **if**-construct has its 'own' set of these variables.

Strictly speaking the program S_{fair} does not belong to $\mathscr{S}_{cn}$ as the **if–then** and the **for**-constructs are not assumed in the syntax. However, it is 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 $\mathscr{M}_{\text{fair}}$ semantics.

Let us call the subprograms introduced in Step 2 the $\mathbf{if}_{\text{fair}}$-constructs. The above transformation boils down to building into all **if**-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}.

Lemma 4.3. *For any program* $S \in \mathscr{S}_{cn}$, $\mathscr{M}_{\text{wfair}}[\![S]\!] = \mathscr{M}_{\text{wtot}}[\![S_{\text{fair}}]\!]$.

Proof. It suffices to prove the following facts:

(a) If ξ is a fair non-failing computation of S, then an extension of it 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.

Ad (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.

Case 1. 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 only been finitely many times enabled when 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.

Case 2. For some state σ_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.

Ad (b). By the construction of S_{fair} the restriction of ξ to the computation steps dealing with S is a non-failing 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. However, this is impossible because as soon as it becomes negative a failure will arise. □

Corollary 4.4. *Suppose that none of the auxiliary variables introduced in* S_{fair} *occurs free in the assertions p and q. Then*

$$\models_{\text{wfair}} \{p\} S \{q\} \quad \textit{iff} \quad \models_{\text{wtot}} \{p\} S_{\text{fair}} \{q\}.$$

Note that we cannot relate the semantics $\mathscr{M}_{\text{fair}}$ and $\mathscr{M}_{\text{tot}}$ in the sense of Lemma 4.3 as for all S and σ we have $\mathbf{fail} \in \mathscr{M}_{\text{fair}}[\![S]\!](\sigma)$. Consequently, using the above

transformation we cannot derive the proof rules for total correctness under the assumption of fairness directly.

4.8. *A proof system dealing with fairness*

The above corollary indicates that in order to prove weak total correctness of S under the assumption of fairness it is sufficient to prove weak total correctness of S_{fair}.

To prove weak total correctness of S_{fair} we can use the proof system CWT defined in Section 4.5.

Assume now for a moment that only deterministic **do**-loops are allowed, i.e., **do**-loops of the form **do** $e \to S$ **od**. Then the first step in the transformation discussed in the previous section is not needed and can be deleted.

Consider now a proof of a correctness formula $\{p_1\} S_{\text{fair}} \{q_1\}$ in the system CWT. Due to the form of S_{fair} any such proof can be transformed into a proof of the correctness formula $\{p_1\} S \{q\}$ provided we use the following transformed version of the **if**-rule:

$$\frac{\{p\}\,\mathbf{if}_{\text{fair}}\,\{q\}}{\{p\}\,\mathbf{if}\ e_1 \to S_1 \,[]\, \cdots \,[]\, e_m \to S_m\ \mathbf{fi}\,\{q\}}.$$

The hypothesis of this rule can be simplified if we 'adsorb' all assignments to the auxiliary variables into the assertion p and apply 'backwards' RULE 4. In such a way we obtain the following proof rule which exclusively deals with the **if**-construct and its original components. Here $\bar{z} \geq 0$ is shorthand for $z_1 \geq 0 \wedge \cdots \wedge z_m \geq 0$.

RULE 11: fair **if**-rule I

$$\frac{\{p\,[\mathbf{if}\ e_j\ \mathbf{then}\ z_j + 1\ \mathbf{else}\ z_j / z_j]_{j \neq i}[1/z_i] \wedge e_i \wedge \bar{z} \geq 0\}\, S_i\, \{q\}_{i=1,\ldots,m}}{\{p\}\ \mathbf{if}\ e_1 \to S_1 \,[]\, \cdots \,[]\, e_m \to S_m\ \mathbf{fi}\,\{q\}}.$$

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 $\mathcal{M}_{\text{wfair}}$ or $\mathcal{M}_{\text{fair}}$ semantics. Therefore, a proof of weak total correctness under the assumption of fairness of the latter program constitutes a proof of weak 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

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

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

Summarizing, the proof system WFN for weak total correctness of programs from $\mathscr{S}_n$ under the assumption of fairness is obtained from the proof system N by replacing the if- and do-rules 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 purpose of introducing the transformation of 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.

4.9. Soundness and completeness of WFN

The following lemma provides a proof-theoretic counterpart of Corollary 4.4.

Lemma 4.5. *Suppose that none of the auxiliary variables introduced in S_{fair} occurs free in the assertions p and q. Then*

$$\mathrm{Tr}_J \vdash_{\mathrm{WFN}} \{p\}S\{q\} \quad \textit{iff} \quad \mathrm{Tr}_J \vdash_{\mathrm{CWT}} \{p\}S_{\text{fair}}\{q\}.$$

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

Lemma 4.5 together with Corollary 4.4 reduces the question of soundness and completeness of WFN to that of CWT. But the latter system is sound and complete in the sense of Section 4.4. This shows that proof system FN is also sound and complete in the same sense. We only have to restrict additionally the class of allowed structures to those which in their **data** domain contain natural numbers.

4.10. The correctness under the assumption of fairness

Proof system WFN is appropriate for proving weak total correctness under the assumption of fairness. To ensure that additionally freedom of failure is guaranteed we proceed in the same way as before—we simply add to the premises of the fair **if**-rule I the assertion $p \to \bigvee_{i=1}^{m} e_i$.

We then obtain the following proof rule:

RULE 13: fair **if**-rule II

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

Replacing in proof system WFN RULE 11 by RULE 13 we obtain a proof system appropriate for proving total correctness under the assumption of fairness. We call

this system FN. This system is sound and complete in the above sense w.r.t. the $\mathcal{M}_{\text{fair}}$ semantics.

Finally we comment on the use of ordinals in RULE 12.

Similarly as in the case of proof system CNT exactly all recursive ordinals are here needed when the **data** domain consists of natural numbers. In [4] it is proved that even if we restrict ourselves to the class of programs disallowing nested nondeterminism, then still the same set of ordinals is needed for proofs carried out in system FN.

4.11. 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 $x > 0 \rightarrow$ **if true** $\rightarrow$ **if** $b \rightarrow x := x - 1$

$\square\ b \rightarrow b :=$ **false**

$\square\ \neg b \rightarrow$ **skip fi**

$\square$ **true** $\rightarrow b :=$ **true fi**

od

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

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

$$p(\alpha) \wedge \alpha > 0 \rightarrow x > 0, \tag{14}$$

$$p(0) \rightarrow x = 0, \tag{15}$$

$$\exists \alpha\, p(\alpha) \tag{16}$$

and

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

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

$$p(\alpha) \Leftrightarrow \alpha = \textbf{if } x > 0 \quad \textbf{then } \rho(x, z_3, 1 - b, (b \rightarrow z_1, z_2))$$

$$\textbf{else } 0.$$

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 first have to prove the premises

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

and

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

as the first premise of the fair **if**-rule is obviously satisfied. Here

$$S_1 \equiv \textbf{if} \quad b \rightarrow x := x - 1$$
$$b \rightarrow b := \textbf{false}$$
$$\neg b \rightarrow \textbf{skip fi}.$$

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

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

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

and

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

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,

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

which implies (20) by the consequence rule 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

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

so (21) by the consequence rule.

Finally, to prove (22) we note that

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

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 **if**-rule.

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

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

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, $\{r\}\ b := \textbf{true}\ \{\exists \beta < \alpha p(\beta)\}$ so (19) by the consequence rule.

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

4.12. The issue of justice

Another possible restricted interpretation of nondeterministic programs is the one under the assumption of justice. In the context of programs from $\mathscr{S}_n$ 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 **true** 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 4.7 the program from the first line in Step 2 by

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

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 4.7 we can omit the conditions $z_i = 0$ from all of the guards, both for the case of fairness and justice. Clearly various other transformations also satisfy Lemma 4.3. We chose here a transformation which leads to simplest proof rules dealing with fairness or justice.

5. Conclusions

In this survey we showed how the issue of correctness of nondeterministic programs can be studied within the framework of Hoare's logic. It seems instructive to provide now a critical assessment of this approach. The remarks below apply both to this and previous part of the survey.

The characteristic feature of all proof systems here considered is that they are *syntax directed* in the sense that the proof rules follow the syntax of the language constructs. This feature of proof systems makes the task of finding a correctness proof of a given program easier and more manageable. What is perhaps even more important is that these proof systems allow to develop programs together with the corresponding correctness proof. Dijkstra [8] provides several convincing examples of such an approach to program design even though he does not use the formalism adopted here. Also the completeness proofs are constructive and provide a heuristic which can be helpful when trying to find concrete proofs.

It should be noted, however, that the proof systems studied here are not completely adequate for proving correctness of the programs in the sense required by the practical considerations.

We considered here only one type of failure due to an evaluation of all guards of an **if**-construct to **false**. In practice, different types of failures can arise like overflow, underflow, stack overflow, division by zero, use of uninitialized variables etc. Most of these failures can be taken care of in a natural way by using appropriately strengthened axioms and proof rules (see, e.g., the note in Section 3.4).

However, not all program properties can be taken care of in such a simple way. For example, the proof rules dealing with fairness are fairly complicated and certainly not easy to use. In the case of concurrent programs various other important properties and hypotheses (see, e.g., [19]) cannot be naturally expressed and axiomatized in Hoare-like logics either.

Hoare's approach was originally concerned with input–output analysis of program behaviour, that is to say, the study of the relation between the input and output states. However, not all program properties are of this type. A finer analysis of the program behaviour requires a study of *execution sequences* (viz. the hypothesis of fairness) and Hoare's logic does not seem an appropriate tool for such a study any more. More appropriate framework for such an analysis seems to be temporal logic (see, e.g., [19]) which explicitly deals with the properties of sequences of states and not states only.

6. Bibliographical remarks

The first treament 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 [8] using a different approach. AXIOMS 1 and 2 and proof RULES 3 and 6 are from Hoare [14]. RULES 4 and 5 are obvious modifications of the appropriate rules dealing with the deterministic versions of the constructs and introduced in [15] and [14], respectively. They appear for example in [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 [5, Section 3]. RULE 7 is inspired by the discussion of clean behaviour of programs in [19]. The completeness proof from Section 3.8 is an appropriate modification of a corresponding proof from [11].

The notion of bounded nondeterminism is introduced in [8]. Countable nondeterminism is extensively studied in [3] and several related references can be found there. Corollary 4.1 is implicit in [8]. AXIOM 9 is from [11]. RULE 10 is from [3] where a slightly different syntax is used. Sections 4.3 and 4.4 are based on [3], as well. The program from Section 4.3 is from [8].

The issue of fairness is discussed in several papers (see, for example, [19]). First proof rules dealing with fairness were proposed in [10, 17, 2]. In [16] a simplified completeness proof of a rule is given, which was introduced in [10]. Sections 4.7–4.12 are based on [4]. Transformations realizing fairness were first introduced in [2]. Other versions of such transformations are given and discussed in [18].

The program studied in Section 4.11 is due to S. Katz. First proof rules dealing with justice were proposed in [3, 17]. In [16] another proof rule for justice is given. In [17] arguments for introducing the hypotheses of justice and fairness when studying parallel programs are given. In [20] a thorough discussion of various possible formalizations of the assumption of fairness is given.

Acknowledgment

We thank the referee for useful comments on the previous version of this paper. S. Arun-Kumar suggested a simpler presentation of proof system FN.

References

[1] K.R. Apt, Ten years of Hoare's logic, a survey—Part I, *TOPLAS* **3** (4) (1981) 431–483.

[2] K.R. Apt and E.-R. Olderog, Proof rules and transformations dealing with fairness, *Sci. Comput. Programming* **3** (1) (1983) 65–100. Extended abstract appeared in: *Logic of Programs*, Lecture Notes in Computer Science **131** (Springer, New York, 1982) pp. 1–8.

[3] K.R. Apt and G.D. Plotkin, Countable nondeterminism and random assignment, Tech. Rept. 82-7, L.I.T.P., Université Paris 7, 1982; extended abstract appeared as: A Cook's tour of countable nondeterminism, in: *Proc. ICALP'81*, Lecture Notes in Computer Science **115** (Springer, Berlin, 1981) pp. 479–494.

[4] K.R. Apt, A. Pnueli and J. Stavi, Fair termination revisited—with delay, in: *Proc. 2nd Conf. on Foundations of Software Technology and Theoretical Computer Science*, Bangalore, India (1982) pp. 146–170.

[5] J.W. De Bakker, *Mathematical Theory of Program Correctness* (Prentice-Hall, Englewood Cliffs, NJ, 1980).

[6] S.A. Cook, Soundness and completeness of an axiom system for program verification, *SIAM J. Comput.* **7** (1) (1978) 70–90.

[7] E.W. Dijkstra, Guarded commands, nondeterminacy and formal derivation of programs, *Comm. ACM* **18** (8) (1975).

[8] E.W. Dijkstra, *A Discipline of Programming* (Prentice-Hall, Englewood Cliffs, NJ, 1976).

[9] R.W. Floyd, Nondeterministic algorithms, *J. ACM* **14** (4) (1967) 636–644.

[10] O. Grümberg, N. Francez, J.A. Makowsky and W.P. de Roever, A proof rule for fair termination of guarded commands, in: J.W. de Bakker and J.C. van Vliet eds., *Algorithmic Languages* (IFIP, North-Holland, Amsterdam, 1981) pp. 399–416.

[11] D. Harel, *First-order Dynamic Logic*, Lecture Notes in Computer Science **68** (Springer, New York, 1979).

[12] D. Harel and D. Kozen, A programming language for the inductive sets, and applications, in: *Proc. ICALP'82*, Lecture Notes in Computer Science **140** (Springer, Berlin, 1982) pp. 313–329.

[13] M.C.B. Hennessy and G.D. Plotkin, Full abstraction for a simple programming language, in: *Proc. 8th Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science **74** (Springer, New York, 1979) pp. 108–120.

[14] C.A.R. Hoare, An axiomatic basis for computer programming, *Comm. ACM* **12** (10) (1969) 576–580, 583.

[15] P.E. Lauer, Consistent formal theories of the semantics of programming languages, Tech. Rept. TR. 25.121, IBM Lab. Vienna, 1971.

[16] D. Lehmann, Another proof for the completeness of a rule for the fair termination of guarded commands and another rule for their just termination, Tech. Rept. IW 178–81, Mathematisch Centrum, 1981.

[17] D. Lehmann, A. Pnueli and J. Stavi, Impartiality, justice and fairness: The ethics of concurrent termination, *Proc. ICALP'81*, Lecture Notes in Computer Science **115** (Springer, Berlin, 1981) pp. 264–277.

[18] D. Park, A predicate transformer for weak fair iteration, in: *Proc. 6th IBM Symp. on Mathematical Foundations of Computer Science*, Hakone, Japan, 1981.

[19] A. Pnueli, The temporal semantics of concurrent programs, *Theoret. Comput. Sci.* **13** (1) (1981) 45–60.

[20] J.P. Queille and J. Sifakis, Fairness and related properties in transition systems—a temporal logic to deal with fairness, *Acta Inform.* **19** (3) (1983) 195–220.

[21] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability* (McGraw-Hill, New York, 1967).