A COOK'S TOUR OF COUNTABLE NONDETERMINISM

(extended abstract) by

K.R. Apt
Faculty of Economics, Erasmus University, Rotterdam

and

G.D. Plotkin
Dept. of Computer Science, University of Edinburgh

ABSTRACT

We provide four semantics for a small programming language involving unbounded (but countable) nondeterminism. These comprise an operational one, two denotational ones based on the Egli-Milner and Smyth orders, respectively, and a weakest precondition semantics. Their equivalence is proved. We also introduce a Hoare-like proof system for total correctness and show its soundness and completeness in an appropriate sense. Admission of countable nondeterminism results in a lack of continuity of various semantic functions; moreover some of the partial orders considered are in general not cpo's and in proofs of total correctness one has to resort to the use of (countable) ordinals. Proofs will appear in the full version of the paper.

1. INTRODUCTION

One of the natural assumptions concerning the execution of a nondeterministic or parallel program is that of fairness. In its simplest form it states that no process is forever denied its turn for execution. The assumption of fairness implies unbounded nondeterminism. To see this, consider the well-known program, b := true; x := 0; do b → x := x + 1 □ b → b := false od (see Dijkstra [8], p. 76), which always terminates, under the assumption of fairness, and assigns to x an arbitrary natural number depending on the sequence of execution steps. What is more, every nondeterministic program of this kind can be translated into an appropriate unbounded nondeterministic program using the random assignment command x := ? which sets x to an arbitrary integer. This close relation between fairness and unbounded (but countable) nondeterminism motivates us to a thorough study of the latter.

As is also well-known, unbounded nondeterminism results in a lack of continuity of various semantic functions. For example, in Dijkstra [8], Ch. 9, one can find an argument showing that admitting unbounded nondeterminism results in a noncontinuity of the weakest precondition semantics. On the other hand, Boom [5] realized that this weakest precondition semantics still can be straightforwardly defined by considering least fixed-points of monotone but non-continuous functions. Both Broy et al [6] and Back [4] gave semantics for unbounded nondeterminism, employing variants of the discrete powerdomains in [15]. The former paper used least fixed-points but the latter (unfortunately) only used the first ω iterates. Similar issues are addressed in Park [14] where the assumption of fair merging is also analysed.

In other papers the issue of complexity of these properties is raised. In particular Chandra [7] has shown that the halting problem for programs admitting unbounded nondeterminism is of higher complexity than truth in the standard model of natural numbers. Similar results concerning various assumptions of fairness and inevitability about simple nondeterministic programs were proved in Emerson and Clarke [9].

In the present paper we try to consider all these issues together, concentrating on a simple programming language with atomic commands allowing countable nondeterminism (such as random assignment). In section 2 we define discrete powerdomains considering both the Egli-Milner ordering and the Smyth ordering, where we no longer obtain a cpo. The section concludes with a systematic presentation of predicate transformers which adapts Dijkstra's healthiness conditions to the present framework and shows the connection with Smyth powerdomains (in analogy with Plotkin [16]). In section 3 we present two denotational semantics, a predicate transformer semantics and an operational one. The relationships between all four are shown. In section 4 we consider a Hoare-style logic for total correctness and present soundness and relative completeness results; this involves the use of countable ordinals in the assertions. In a fuller version of the paper we would like to integrate Chandra's ideas on computability into our framework.

What we have shown here is that unbounded nondeterminism admits a simple and natural characterization which can be studied by generalizing techniques used for the case of deterministic or bounded nondeterministic programs.

The present work can be easily extended to cover some other constructs omitted in our analysis such as or commands, Dijkstra's guarded commands or recursive procedures. For example, the proof system we consider is a simple refinement of the corresponding system for total correctness of while programs and an appropriate system covering the case of recursion is a similar refinement of a system dealing with total correctness of recursive procedures (see for example Apt [1]).

In principle our paper also provides a framework for studying fairness via translation into a language for countable nondeterminism. A proof theoretic approach to the problem of total correctness of fair nondeterministic programs based on this idea has been recently worked out in Apt and Olderog [2]. Even though such methods are operational in nature, they turn out to be natural and easy to apply in practice.

2. POWERDOMAINS AND PREDICATE TRANSFORMERS

In this section we gather some general information on fixed-points that we will need later. Then we give the basic definitions and properties of discrete powerdomains, suitably adapted from these in Plotkin [16] and Smyth [17] to handle countable non-determinism. Finally we consider adapting the predicate transformers in Dijkstra [8] to handle countable non-determinism and show, following the ideas in Plotkin [16], how they connect up with the discrete Smyth powerdomain.

Definition 2.1 Let P be a partial order and let A be subset of P. Then A is directed if every finite subset of A has an upper bound in A; it is countably directed (ω-directed) if every countable subset of A has an upper bound in A. The partial order P is a cpo (complete partial order) if every directed subset, A, of P has a lub (least upper bound), denoted by $\bigsqcup A$, and if P has a least element, denoted by $\bot$. A subset of P is eventually constant if it contains its own least upper bound.

For example for any set, X, there is the flat cpo $X_\bot$ which is the set $X \uplus \{\bot\}$ ordered by : $x \sqsubseteq y$ iff $x = \bot$ or $x = y$.

Definition 2.2 Let P, Q be partial orders and let $f : P \to Q$ be a monotone function. Then f is continuous if whenever $A \subseteq P$ is a directed subset with a lub, then f(A) has a lub, namely $f(\bigsqcup A)$ (i.e. f preserves lubs of directed subsets); f is strict whenever it preserves the least element.

Definition 2.3 Let P, Q be partial orders, X a countable set. Then $P \times Q$ is the Cartesian product of P and Q ordered coordinatewise; $X \to P$ is the partial order of all functions from X to P ordered pointwise.

Fact 2.1 If P is a cpo then so is $X \to P$; if P and Q are cpo's so is $P \times Q$.

Fixed Points For any partial order P, any monotone $f : P \to P$ and all ordinals λ, define f^λ by:

$$f^{\lambda} = f\left(\bigsqcup_{k < \lambda} f^{k}\right)$$

Of course f^{λ} need not exist since $\bigsqcup_{k < \lambda} f^{k}$ need not exist. If f^{λ} does not exist then for any $\lambda' > \lambda$, $f^{\lambda'}$ does not exist either; f^{λ} is monotone in λ. If $\{f^{\lambda}\}_{\lambda}$ stabilizes at k, then f^{k} is the least (pre-)fixed-point of f. If P is a cpo then f^{λ} always exists and $\{f^{\lambda}\}_{\lambda}$ stabilizes. If additionally f is continuous then $\{f^{\lambda}\}_{\lambda}$ stabilizes at ω.

Discrete Powerdomains

We explore Egli-Milner and Smyth powerdomains of flat cpo's, X, with enough subsets to handle countable nondeterminism. To avoid some ticklish problems we restrict X to being countable. Note that, even so, the Smyth powerdomain is not a cpo; we do not understand what significance this has for a possible more general theory of powerdomains for countable nondeterminism.

Egli-Milner Order

Let $\mathcal{E}(X_{\perp})$ be the set of non-empty subsets of X ordered by:

$$A \sqsubseteq B \text{ iff } (\forall a \in A.\ \exists b \in B.\ a \sqsubseteq b) \wedge (\forall b \in B.\ \exists a \in A.\ a \sqsubseteq b)$$

(which is the same as $A = B$ (if $\perp \notin A$) or as $A - \{\perp\} \subseteq B$ (if $\perp \in A$)).

Proposition 2.1 $\mathcal{E}(X_{\perp})$ is a cpo with least element $\{\perp\}$; every ω-directed subset is eventually constant; it is closed under arbitrary unions. ⊠

Useful Functions

Singleton $\{\cdot\} : X \to \mathcal{E}(X_{\perp})$

Union $\cup : \mathcal{E}(X_{\perp})^2 \to \mathcal{E}(X_{\perp})$. It is continuous.

Extension For $f : X \to \mathcal{E}(Y_{\perp})$ define $f^{+} : \mathcal{E}(X_{\perp}) \to \mathcal{E}(Y_{\perp})$ by:
$f^{+}(A) = \cup f(A - \{\perp\}) \cup \{\perp \mid \perp \in A\}$

Proposition 2.2 Every f^{+} is continuous. However, f^{+} is not continuous as a functions of f although it is monotonic. ⊠

Composition For $X \xrightarrow{f} \mathcal{E}(Y_{\perp})$, $Y \xrightarrow{g} \mathcal{E}(Z_{\perp})$ define $X \xrightarrow{f;g} \mathcal{E}(Z_{\perp})$ by:

$$f ; g = g^{+} \circ f$$

Proposition 2.3 The composition f;g is continuous in f and monotonic, but not continuous in g. Also it is associative with units the singleton functions (i.e. we get a category).

Note It is the lack of continuity of f;g in g that will force us (in the semantics of while commands) to consider least fixed-points of non-continuous functionals.

Smyth Order

Let $\mathcal{U}(X_\perp)$ be

$$\{A \subseteq X \mid A \neq \emptyset\} \cup \{X_\perp\}$$

ordered by the superset ordering:

$$A \sqsubseteq B \text{ iff } A \supseteq B$$

(for motivations for this definition see Plotkin [16].

Proposition 2.4 $\mathcal{U}(X_\perp)$ has least element $X_\perp$ but need not be a cpo (although if $\mathfrak{I}$ has an upper bound, its lub exists and is $\cap\mathfrak{I}$); every ω-directed subset is eventually constant; it is closed under arbitrary unions. ⊠

Note Greatest lower bounds of non-empty families, $\mathfrak{I}$, always exist being given by: $\sqcap\mathfrak{I} = \cup\mathfrak{I}$.

Useful Functions

Singleton $\{\cdot\} : X \to \mathcal{U}(X_\perp)$

Union $\cup : \mathcal{U}(X_\perp)^2 \to \mathcal{U}(X_\perp)$. It is continuous.

Extension For $f : X \to \mathcal{U}(Y_\perp)$ define $f^+ : \mathcal{U}(X_\perp) \to \mathcal{U}(Y_\perp)$ by:

$$f^+(A) = \begin{cases} \cup f(A) & (\perp \notin A) \\ Y_\perp & (\perp \in A) \end{cases}$$

Proposition 2.5 Every f^+ is monotone, but not necessarily continuous; function extension, $(\cdot)^+$ is monotonic, but not necessarily continuous. ⊠

Composition For $X \xrightarrow{f} \mathcal{U}(Y_\perp)$ and $Y \xrightarrow{g} \mathcal{U}(Z_\perp)$ define $X \xrightarrow{f;g} \mathcal{U}(Z_\perp)$ by:

$$f;g = g^+ \circ f$$

Proposition 2.6 The composition f;g is monotonic in each argument, but need not be continuous in either. Also it is associative with the singleton as unit. ☒

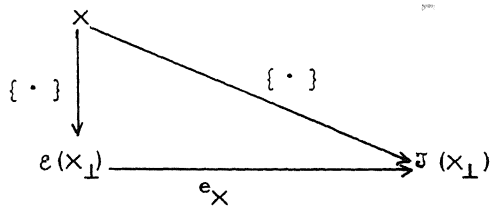
From $\mathcal{E}(X_\perp)$ to $\mathcal{T}(X_\perp)$

Define e_X: $\mathcal{E}(X_\perp) \to \mathcal{T}(X_\perp)$ by:

$$e_X(A) = \begin{cases} A & (\perp \notin A) \\ X_\perp & (\perp \in A) \end{cases}$$

(That is, $e_X(A) = \{b \in X_\perp \mid \exists\, a \in A.\ a \sqsubseteq b\}$).
Then e_X is strict and continuous. It is very important that e_X is continuous as this is why we can live with the fact that $\mathcal{T}(X_\perp)$ is not a cpo - enough directed sets, for our purpose, will have limits as they will be images under e_X of directed sets in $\mathcal{E}(X_\perp)$.

Fact 2.2 The following diagram commutes:



☒

Fact 2.3 For any $f : X \to \mathcal{E}(Y_\perp)$ and $g : \to \mathcal{E}(Z_\perp)$, $e_Z \circ (f;g) = (e_Y \circ f);(e_Z \circ g)$. ☒

Smyth Powerdomains and Predicate Transformers

A predicate transformer from X to Y is any map $p : \mathcal{P}(Y) \to \mathcal{P}(X)$ such that:

(1) Law of Excluded Miracle $p(\emptyset) = \emptyset$
(2) Countable Multiplicativity $p(\bigcap_{i \in \omega} B_i) = \bigcap_{i \in \omega} p(B_i)$

These are the appropriate healthiness conditions. The usual healthiness conditions imply them (recall here that X, Y are taken as countable) but non-continuous transformers are allowed - and as is, essentially, pointed out in Dijkstra [8], Ch. 9, must be. That they are exactly the right conditions will appear from the isomorphism with the Smyth powerdomain functions that we will show and from the role they play in the various semantics.

We take $PT_{X,Y}$ to be the set of predicate transformers from X to Y (dropping the subscripts when they can be understood from the context) and ordered pointwise thus:

$$p \subseteq q \text{ iff } \forall B \subseteq Y.\ p(B) \subseteq q(B)$$

The "Smyth state transformers" from X to Y are all functions $m : X \to \mathcal{P}(Y_\perp)$, also ordered pointwise: this collection is called $ST_{X,Y}$. Now for any such m define for $B \subseteq Y$:

$$wp(m, B) = \{a \in X \mid m(a) \subseteq B\}$$

Note If $\perp \in m(a)$ then never $a \in wp(m, B)$.

Lemma 2.1 The function $wp(m, \cdot)$ is a predicate transformer and $wp(m, \cdot)$ is monotonic in m. ⊠

So now we have a monotonic $\omega : ST \to PT$ where $\omega(m)(B) = wp(m, B)$, and the next theorem even shows it is an isomorphism.

Theorem 2.1 (Isomorphism) The function $\omega : ST \tilde{=} PT$ is an isomorphism of partial orders. ⊠

3. SEMANTIC ISSUES

In this section we consider four semantics of a simple programming language of commands allowing countable nondeterminism and establish the relationships between the various semantics. The first semantics is operational being given as a transition relation between configurations and specified axiomatically. The next two are standard denotational semantics based on the two discrete powerdomains we consider in section 2. The last is a denotational predicate transformer semantics.

We disagree with Back [4] who defines a semantics also based on $\mathcal{E}(X_\perp)$ but different from ours in that the semantics of while-loops is defined as the limit of the first ω-iterates. He correctly points out that this does not capture the correct notion of termination and blames that on a failure of $\mathcal{E}(X_\perp)$; we rather blame it on the semantics he gives to while-loops and prefer to carry the iterates to enough stages (at most all countable ordinals) to reach the least fixed-point as in [6]. Then with this definition, theorem 3.1 below shows the operational and denotational semantics are identical.

Further fact 3.1 shows the semantics based on the Smyth order is a projection, under e_X, of the semantics based on the Egli-Milner ordering and corollary 3.1 then relates it to the operational semantics. Finally we give a predicate transformer semantics, again iterating through suitable ordinals, following Boom [5], and show in

theorem 3.2 and corollary 3.2 that it is isomorphic to the semantics based on the Smyth order (following the ideas in Plotkin [16]).

Throughout the rest of the paper we consider a simple programming language whose set of commands is parameterised on two sets:

ACom is the set of atomic commands ranged over by the metavariable A.
BExp is the set of Boolean expressions ranged over by B.

Now, Com is the set of commands of the language, ranged over by S and generated by the following grammar:

$$S ::= \underline{skip} \mid A \mid S;S \mid \underline{if}\ B\ \underline{then}\ S\ \underline{else}\ S\ \underline{fi} \mid \underline{while}\ B\ \underline{do}\ S\ \underline{od}$$

We assume a countable unanalysed set X of states and we further assume we are given two semantic functions:

$$\mathcal{A} : ACom \to (X \to \mathcal{P}(X) - \{\emptyset\})$$
$$\mathcal{B} : BExp \to (X \to \{tt, ff\})$$

where {tt, ff} is of course the set of truthvalues.

The assumption that for any $\sigma \in X$, $\mathcal{A}[\![A]\!](\sigma)$ is a non-empty and (necessarily) countable subset of X means that atomic commands are assumed to be always terminating and countably nondeterministic statements. A particular choice for A might be the statement $x := ?$, meaning set x to any value. If there were only one variable that could appear in the language we could give the semantics of $x := ?$ by putting for any σ:

$$\mathcal{A}[\![x := ?]\!](\sigma) = X$$

We now provide three different semantics for commands.

Operational Semantics

We define a function

$$Op : Com \to (X \to \mathcal{E}(X_\perp))$$

by considering a transition relation " $\to$ " between configurations, that is pairs $\langle S, \sigma \rangle$ consisting of a command and a state. We define " $\to$ " by the following clauses:

I. $\langle A, \sigma\rangle \to \langle \underline{skip}, \sigma'\rangle$ if $\sigma' \in \mathcal{G}[\![A]\!](\sigma)$

II. If $\langle S_1, \sigma\rangle \to \langle S_1', \sigma'\rangle$ then $\langle S_1; S, \sigma\rangle \to \langle S_1'; S, \sigma'\rangle$

III. $\langle \underline{skip}; S, \sigma\rangle \to \langle S, \sigma\rangle$

IV. 1. $\langle \underline{if}\ B\ \underline{then}\ S_1\ \underline{else}\ S_2\ \underline{fi}, \sigma\rangle \to \langle S_1, \sigma\rangle$ (if $\mathcal{B}[\![B]\!](\sigma) = tt$)

2. $\langle \underline{if}\ B\ \underline{then}\ S_1\ \underline{else}\ S_2\ \underline{fi}, \sigma\rangle \to \langle S_2, \sigma\rangle$ (if $\mathcal{B}[\![B]\!](\sigma) = ff$)

V. 1. $\langle \underline{while}\ B\ \underline{do}\ S\ \underline{od}, \sigma\rangle \to \langle S; \underline{while}\ B\ \underline{do}\ S\ \underline{od}, \sigma\rangle$ (if $\mathcal{B}[\![B]\!](\sigma) = tt$)

2. $\langle \underline{while}\ B\ \underline{do}\ S\ \underline{od}, \sigma\rangle \to \langle \underline{skip}, \sigma\rangle$ (if $\mathcal{B}[\![B]\!](\sigma) = ff$)

Intuitively, $\langle S_1, \sigma\rangle \to \langle S_2, \sigma'\rangle$ means that one step of execution of S_1 in state σ can lead to state σ' with S_2 being the remainder of S_1 to be executed.

Definition 3.1 S can diverge from σ iff there exists an infinite sequence $\langle S_i, \sigma_i\rangle$ $(i = 0, 1 \ldots\ldots)$ such that $\langle S, \sigma\rangle = \langle S_0, \sigma_0\rangle \to \langle S_1, \sigma_1\rangle \to \langle S_2, \sigma_2\rangle \to \ldots$

Note a) If $S \neq \underline{skip}$ then for any σ there are S' and σ' such that $\langle S, \sigma\rangle \to \langle S', \sigma'\rangle$ (that is, S can be executed for at least one step)

b) The set $\{\langle S', \sigma'\rangle \mid \langle S, \sigma\rangle \to \langle S', \sigma'\rangle\}$ is always countable (since X is assumed to be countable).

Definition 3.2 We define the function Op by:

$$Op[\![S]\!](\sigma) = \{\sigma' \mid \langle S, \sigma\rangle \to^* \langle \underline{skip}, \sigma'\rangle\} \cup \{\perp \mid S \text{ can diverge from } \sigma\}.$$

Of course "$\to^*$" is the transitive reflexive closure of "$\to$".

Denotational Semantics

We define now two functions

$$\mathcal{S}_{\mathcal{E}}: \text{Com} \to (X \to \mathcal{E}(X_\perp))$$

and

$$\mathcal{S}_{\mathcal{T}}: \text{Com} \to (X \to \mathcal{T}(X_\perp))$$

by the same type of equations. Let $\mathcal{N}$ be, indifferently, $\mathcal{E}$ or $\mathcal{T}$. We define

I. $\mathcal{S}_{\mathcal{N}}[\![\underline{skip}]\!] = \{\cdot\}$ ($\{\cdot\}$ is the singleton function defined in section 2)

II. $\mathcal{S}_{\mathcal{N}}[\![A]\!] = \lambda\sigma \in X.\ \mathcal{G}[\![A]\!](\sigma)$

III. $\mathcal{S}_{\mathcal{N}}[\![S_1; S_2]\!] = \mathcal{S}_{\mathcal{N}}[\![S_1]\!]; \mathcal{S}_{\mathcal{N}}[\![S_2]\!]$

IV. $\mathcal{S}_{\mathcal{N}}[\![\underline{if}\ B\ \underline{then}\ S_1\ \underline{else}\ S_2\ \underline{fi}]\!](\sigma) = \underline{if}\ \mathcal{B}[\![B]\!](\sigma)\ \underline{then}\ \mathcal{S}_{\mathcal{N}}[\![S_1]\!](\sigma)\ \underline{else}\ \mathcal{S}_{\mathcal{N}}[\![S_2]\!]$

V. $\mathcal{S}_{\mathcal{N}}[\![\underline{while}\ B\ \underline{do}\ S\ \underline{od}]\!] = \mu m.\ \lambda\sigma \in X.\ \underline{if}\ \mathcal{B}[\![B]\!](\sigma)\ \underline{then}\ (\mathcal{S}_{\mathcal{N}}[\![S]\!]; m)(\sigma)\ \underline{else}$.

Note $\mathcal{T}(X)$ need not be a cpo, so $\mathcal{D}_{\mathcal{T}}$ might be not well-defined in case V. But this is not the case because of the following fact which also shows the relationship between the two denotational semantics.

Fact 3.1 For all S, $\mathcal{D}_{\mathcal{T}}$ is well-defined and $e_X \circ \mathcal{D}_{\mathcal{C}}[\![S]\!] = \mathcal{D}_{\mathcal{T}}[\![S]\!]$. ⊠

The equivalence of the denotational and operational semantics is expressed in the following theorem:

Theorem 3.1 $\mathcal{D}_{\mathcal{C}} = Op$. ⊠

Corollary 3.1 (Operational characterisation of $\mathcal{D}_{\mathcal{T}}$)

i) If S cannot diverge from σ then
$\sigma' = \mathcal{D}_{\mathcal{T}}[\![S]\!](\sigma)$ iff $\langle S, \sigma\rangle \rightarrow^* \langle \underline{skip}, \sigma'\rangle$

ii) $\perp \in \mathcal{D}_{\mathcal{T}}[\![S]\!](\sigma)$ iff S can diverge from σ

Proof By fact 3.1 and theorem 3.1. ⊠

Weakest Precondition Semantics

Let PT be the set of all predicate transformers from X to X as defined in section 2. We define now a function $\wp : Com \rightarrow PT$ which we shall call the weakest precondition semantics (wp semantics).

I. $\wp[\![\underline{skip}]\!] = id$

II. $\wp[\![A]\!](R) = wp(\mathcal{G}[\![A]\!], R)$ (where wp is the function defined in section 2).

III. $\wp[\![S_1; S_2]\!] = \wp[\![S_1]\!] \circ \wp[\![S_2]\!]$

IV. $\wp[\![\underline{if}\ B\ \underline{then}\ S_1\ \underline{else}\ S_2\ \underline{fi}]\!](R) = (\mathcal{B}[\![B]\!]^{-1}(tt) \cap \wp[\![S_1]\!](R)) \cup (\mathcal{B}[\![B]\!]^{-1}(ff) \cap \wp[\![S_2]\!](R))$

V. $\wp[\![\underline{while}\ B\ \underline{do}\ S\ \underline{od}]\!](R) =$
$\mu Q \subseteq X.((\mathcal{B}[\![B]\!]^{-1}(tt) \cap \wp[\![S]\!](Q)) \cup (\mathcal{B}[\![B]\!]^{-1}(ff) \cap R))$

It is clear that $\wp$ is well-defined, as $\wp[\![S]\!]$ is monotone and so the corresponding function in case V is monotone as well, and therefore has a least fixed-point. However, we also wish to prove that for each S, $\wp[\![S]\!]$ is a predicate transformer. This follows directly from the next theorem which also establishes the relationship with the semantics based on the Smyth powerdomain.

Theorem 3.2 For all $S \in Com$ and $R \subseteq X$ we have:

$$wp(\mathcal{D}_{\mathcal{T}}[\![S]\!], R) = \wp[\![S]\!](R) \quad ⊠$$

Corollary 3.2 For all S in Com and $R \subseteq X$ we have:

$$\mathfrak{D}_{\mathcal{T}} [\![S]\!] = \omega^{-1} (\nu [\![S]\!])$$

Proof By theorems 3.2 and 3.1. ⊠

Corollary 3.3 (Operational characterisation of wp semantics)

$\sigma \in \nu [\![S]\!] (R)$ iff S cannot diverge from σ and
$\forall \sigma'. [\langle S, \sigma\rangle \to^* \langle \underline{skip}, \sigma'\rangle \to \sigma' \in R]$

Proof By corollary 3.1 and theorem 3.2. ⊠

4. PROOF THEORY

In this section we consider a Hoare logic for the total correctness of programs and indicate the soundness of the logic and a relative completeness theorem after the fashion of Cook (see Apt [1] for a survey of results of this kind). As our assertion language, L, we take any many-sorted logic whose sort set contains a sort data (for program data) and ord (for ordinals); we also assume a constant 0, of sord ord, and a binary predicate symbol, $<$, over ord. We use x, y, z as variables of sort data and α, β, γ as variables of sort ord; we use p, q, r to range over L-formulae.

Now we can finish specifying the syntax of our programming language. For convenience we will only consider a fixed finite set of data variables, $Var = \{x_1 \ldots, x_k\}$. Boolean expressions are taken to be those quantifier-free L-formulae whose variables are all in Var and whose symbols have sorts only involving data. Let t range over expressions of sort data whose symbols have sorts only involving data. Atomic commands are taken to be of the form $x := t$ (ordinary assignment) or $x := ?$ (random assignment).

Before turning to semantic issues we give our logic and work out an example. The formulae of the logic are all L-formulae together with all those of the form

$$\{p\}\ S\ \{q\}$$

(the latter meaning that, for all values of parameters, if σ is a state satisfying p, then every execution sequence of S from σ terminates and ends in a state satisfying q). The axioms and rules of the logic are as follows:

1. Assignment

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

where $p[t/x]$ is the result of substituting t for all free occurrences of x in p.

2. Random Assignment

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

provided x is not free in p.

3. If-Then-Else Rule

$$\frac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \underline{\text{if}}\ B\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}}\ \{q\}}$$

4. Composition Rule

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

5. While Rule

$$\frac{p(\alpha) \wedge 0 < \alpha \rightarrow B,\ \{p(\alpha)\}\ S\ \{\exists \beta < \alpha .\ p(\beta)\},\ p(0) \rightarrow \neg B}{\{\exists \alpha .\ p(\alpha)\}\ \underline{\text{while}}\ B\ \underline{\text{do}}\ S\ \underline{\text{od}}\ \{p(0)\}}$$

We call $p(\alpha)$ the loop invariant.

6. Consequence Rule

$$\frac{p \rightarrow p',\ \{p'\}\ S\ \{q'\},\ q' \rightarrow q}{\{p\}\ S\ \{q\}}$$

Call the above proof system T; we write $F \vdash_T \{p\}\ S\ \{q\}$ to mean that $\{p\}\ S\ \{q\}$ can be proved in T from the formulae in F. The above while rule is a straightforward generalization of the following while rule for total correctness of the usual while programs given in Harel [10].

7. While Rule II

$$\frac{p(\alpha+1) \rightarrow B,\ \{p(\alpha+1)\}\ S\ \{p(\alpha)\}, p(0) \rightarrow B}{\{\exists \alpha .\ p(\alpha)\}\ \underline{\text{while}}\ B\ \underline{\text{do}}\ S\ \underline{\text{od}}\ \{p(0)\}}$$

(A slightly different vocabulary is assumed here, viz. α ranges over the natural numbers). We shall show in a moment that while rule II is not sufficient for proofs of total correctness of programs.

As an example proof in T consider the following program:

S = **while** B **do** S_0 **od**, where

$$B \equiv x = 0 \vee 0 < y$$

and

$S_0 \equiv$ **if** $x = 0$ **then** $y := ?$; $x := 1$ **else** $y := y - 1$ **fi** (see Dijkstra [8], Ch. 9).

We now wish to prove in T that $\{true\}\ S\ \{y = 0\}$ holds. To this end we assume L contains equality symbols of all sorts, the language of Peano arithmetic (and we use $x < y$ as an abbreviation), a one-argument (conversion) function $\bar{\cdot}$ of sort (data, ord) and a constant ω of sort ord.

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 started in σ.

Then $p(\alpha)$ satisfies the premises of the **while** rule so $\{\exists\alpha.\, p(\alpha)\}\ S\ \{p(0)\}$ holds. Also both $\exists\alpha.\, p(\alpha)$ and $p(0) \rightarrow y = 0$ hold, so by the consequence rule $\{true\}\ S\ \{y = 0\}$ holds.

Note **While** rule II is not sufficient to prove the formula $\{true\}\ S\ \{y = 0\}$ from arithmetical assumptions.

The use of parameterized loop invariants combines the technique of using loop invariants and loop counters. The **while** rule II uses integer-valued loop counters as opposed to the **while** rule from T which uses ordinal-valued loop counters. The insufficiency of integer-valued loop counters to prove the above formula $\{true\}\ S\ \{y = 0\}$ was first observed by Back [3]. The use of ordinal-valued loop counters was in fact proposed already in Floyd [10]. In the proof-theoretic framework it was first incorporated in Manna and Pnueli [13] where so-called convergence functions with a range being a well-founded set are used. In the framework of weakest precondition semantics the use of ordinal-valued loop counters was advocated in Boom [5].

We now pass to the problem of soundness and completeness of T and consider interpretation, I, of L. These are ordinary many-sorted structures of the appropriate type, but subject to the following three conditions:

1. The domain, $I_{\underline{data}}$, of sort data is countable.
2. The domain, $I_{\underline{ord}}$, of sort ord is an initial segment of the ordinals.
3. The constant, $\overline{0}$, denotes the least ordinal and the relation symbol, $<$, denotes the strict ordering of the ordinals, restricted to $I_{\underline{ord}}$.

Let us fix on such an interpretation I and finish specifying the semantics of our programming language. The set of states is:

$$X = Var \to I_{\underline{data}}$$

where $I_{\underline{data}}$ is the domain of sort data. Let π range over maps from all L-variables, other than those in Var, to elements of I-domains of the appropriate sort. We write:

$$I \models_{\pi,\sigma} p$$

to mean that p is true in I when the free variables of p denote the values specified by π and σ; we write $I \models_{\pi} p$ for $\forall \sigma.\ I \models_{\pi,\sigma} p$. The definition of $\mathcal{B}$: $BExp \to (X \to \{tt, ff\})$ is now obvious and for $\mathcal{S}$ we have:

$$\mathcal{G}[\![x := t]\!](\sigma) = \{\sigma[I[\![t]\!](\sigma)/x]\}$$

using an obvious notation and

$$\mathcal{G}[\![x = ?]\!](\sigma) = \{\sigma' \mid \exists i \in I_{\underline{data}} .\ \sigma' = \sigma[i/x]\}$$

Now all four semantics considered in the previous section are at our disposal; we concentrate on the weakest precondition semantics, $\wp$. For the truth of Hoare assertions we put for any p, π:

$$[p]_{\pi} = \{\sigma \mid I \models_{\pi,\sigma} p\}$$

and then:

$$\models_{I} \{p\}\ S\ \{q\} \text{ iff} \forall \pi .\ [p]_{\pi} \subseteq \wp[\![S]\!][q]_{\pi} .$$

By corollary 3.3 this is the same as

$$\models_{I} \{p\}\ S\ \{q\} \text{ iff} \forall \pi, \sigma\ (\sigma \in [p]_{\pi} \to (S \text{ cannot diverge from } \sigma \wedge (\forall \sigma' .\ \langle S, \sigma\rangle \to^{*} \langle \underline{skip}, \sigma'\rangle \to \sigma' \in [q]_{\pi})))$$

which is the usual definition of total correctness. We set Tr_I to be the set of all sentences true in I.

Soundness Theorem For any formulae p, q of L and command S if $Tr_I \vdash \{p\}\ S\ \{q\}$ then $\models_{I} \{p\}\ S\ \{q\}$. ⊠

We now state a completeness theorem for assertion languages of a special form; let L include second order set variables a, b, c, ... Set variables are of arbitrary arity. We write $p(a_1, \ldots, a_m, z_1, \ldots, z_n)$ to denote that $a_1, \ldots, a_m, z_1, \ldots, z_n$ are all among free variables of p. The set variables cannot be quantified over.

However, they can be bound by the least fixed-point operator: for any formulae $p(a, x_1, \ldots, x_k)$ where a is a k-ary set variable which always occurs positively in p, $\mu a.p$ is also a formula. (Here a variable always occurs positively in a formula if none of its occurrences in a disjunctive normal form of the formula are in the scope of a negation sign.) $\mu a.p$ has one free variable less than p (a is bound in $\mu a.p$) and gets the following meaning:

$$I \models_{\pi,\sigma} \mu a.p \text{ iff } I \models_{\pi} [R/a], \sigma^p$$

where $R = \mu Q \subseteq (I_{\underline{data}})^k . [I \models_{\pi} [Q/a] (p \leftrightarrow (x_1, \ldots, x_k) \in a)]$. For our interpretation I we now impose the following two additional conditions:

4. The domains of each of the set sorts contain all sets of the appropriate kind.
5. The domain $I_{\underline{ord}}$ consists of all countable ordinals.

We are now in position to state the completeness theorem.

Completeness Theorem For any command S and formulae p, q if $\models_I \{p\}\ S\ \{q\}$ then $Tr_I \vdash_T \{p\}\ S\ \{q\}$. ⊠

The assertion language we have used here is based on the μ-calculus of Hitchcock and Park [12]. It would be interesting to establish what strength of assertion language is really needed for the completeness theorem.

Acknowledgements

This work was carried out with the aid of a Science Research Council grant. We are grateful to A. de Bruin and the other referees for detailed comments; they will be incorporated in the full version of the paper.

REFERENCES

[1] Apt, K.R., Ten Years of Hoare's Logic, A Survey, Part I Fac. of Economics, Univ. of Rotterdam, Technical Report (to appear in TOPLAS), 1979

[2] Apt, K.R. and Olderog, E.R., Proof Rules Dealing With Fairness Bericht Nr. ɯɯ, Inst. Inf. Prakt. Math., Univ. of Kiel (1981)

[3] Back, R.J., Proving Total Correctness of Non-Deterministic Programs in Infinitary Logic, Computing Centre, Univ. of Helsinki, Research Report No. 9 (to appear in Acta Informatica), 1979

[4] Back, R.J., Semantics of Unbounded Non-Determinism, in: Proc. 7th Colloquium Automata, Languages and Programming, Lecture Notes in Computer Science 85, Springer-Verlag, pp. 51-63, 1980

[5] Boom, H.J., A Weaker Precondition for Loops Mathematisch Centrum Report IW 104/78, 1978

[6] Broy, M., Gratz, R. and Wirsing, M., Semantics of Non-Deterministic and Non-Continuous Constructs in Bauer, F.L. and Broy, M. (eds.) Program Construction, International Summer School Marktoberdorf, July 1978 Lecture Notes in Computer Science 69, Springer-Verlag, pp. 553-591, 1979

[7] Chandra, A., Computable Non-Deterministic Functions,in: Proc 19th Annual Symposium on Foundations of Computer Science, pp. 127-131, 1978

[8] Dijkstra, E.W., A Discipline of Programming, Prentice-Hall, 1976

[9] Emerson, E.A. and Clarke, E.M., Characterizing Correctness Properties of Parallel Programs Using Fixpoints, in: Proc 7th Colloquium Automata, Languages and Programming, Lecture Notes in Computer Science 85, Springer-Verlag, pp. 169-181, 1980

[10] Floyd, R.W., Assigning Meanings to Programs, in: Proc. AMS Symposium in Applied Mathematics 19, pp. 19-31, 1967

[11] Harel, D., First-Order Dynamic Logic, Lecture Notes in Computer Science 68, Springer-Verlag, 1979

[12] Hitchcock, P., Park, D., Induction Rules and Termination Proofs, in: Automata, Languages and Programming (ed. M. Nivat) North Holland, 1973

[13] Manna, Z. and Pnueli, A., Axiomatic Approach to Total Correctness of Programs, Acta Informatica 3, pp. 253-262, 1974

[14] Park, D., On The Semantic of Fair Parallelism, in: Proc. Winter School on Formal Software Specification, Lecture Notes in Computer Science 86, Springer-Verlag, pp. 504-526, 1980

[15] Plotkin, G.D., A Powerdomain Construction, SIAM Journal on Computation Vol. 5, No. 3, pp. 452-487, 1976

[16] Plotkin, G.D., Dijkstra's Predicate Transformer and Smyth's Powerdomain in: Proc. Winter School on Formal Software Specification, Lecture Notes in Computer Science 86, Springer-Verlag, pp. 527-553, 1980

[17] Smyth, M., Powerdomains,JCSS, Vol. 16, No. 1, 1978