stichting
mathematisch
centrum

AFDELING INFORMATICA IW 196/82 MAART
(DEPARTMENT OF COMPUTER SCIENCE)

J.W. DE BAKKER, J.-J.Ch. MEYER & J.I. ZUCKER

ON INFINITE COMPUTATIONS IN DENOTATIONAL SEMANTICS

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

1980 Mathematics subject classification: 68B10, 68C05

1982 CR. Categories: F3.2, F3.3

On infinite computations in denotational semantics*)

by

J.W. de Bakker, J.-J.Ch. Meyer**) & J.I. Zucker***)

ABSTRACT

Finite and, especially, infinite computations in languages with iteration or recursion are studied in the framework of denotational semantics, and a theorem is proved which relates their syntactic and semantic characterizations. A general proof method is presented to establish this type of relations, and it is shown how - in an induction on the structure of the syntactic constructs of the language - the recursive case follows from the non-recursive one by applying a general definitional scheme. The method is applicable to a variety of other problems concerning recursive constructs such as, for example, fixed point characterizations of several notions of weakest precondition. Also, the connections with the theory of languages with infinite words are discussed, in particular with a substitution theorem due to Nivat.

KEY WORDS & PHRASES: *infinite computations, denotational semantics, infinite words, recursion, nondeterminacy, weakest preconditions*

---

*) This report will be submitted for publication elsewhere.

**) Department of Computer Science, Free University, De Boelelaan 1081, 1007 MC Amsterdam

***) Department of Mathematics and Computer Science, Bar-Ilan University, Ramat Gan, Israel

1. INTRODUCTION

We study finite and, especially, infinite computations in the framework of denotational semantics, and prove a theorem which relates their syntactic and semantic characterizations. We consider a simple language with as main concepts assignments, composition, some form of iteration or recursion, and nondeterminacy. Let S be any statement in this language. As usual in denotational semantics, its meaning is a mapping from (input) states to sets of (output) states (*sets* because of nondeterminacy). Let "$\perp$", by convention, be the state which is delivered by a nonterminating computation. In general, for any S and input state σ, the set of output states τ consists of a so-called *finite* part - all states $\sigma' \in \tau$ which are $\neq \perp$ - and an *infinite* part, viz. $\{\perp\}$ in case S has at least one nonterminating computation and $\emptyset$ (the empty set) otherwise. For example, for the statement (x:=0) $\cup$ (x:=1) $\cup$ <u>while</u> <u>true</u> <u>do</u> <u>skip</u> <u>od</u> (with "$\cup$" denoting nondeterministic choice) and input σ, the finite part of the output is $\{\sigma\{0/x\},\sigma\{1/x\}\}$, i.e., the state σ with x set to 0 or 1, and the infinite part is $\{\perp\}$. A first result of our paper is a *syntactic* characterization, for each S, of those computations which deliver the finite and infinite parts of the output, respectively. More specifically, we introduce mappings *fin* and *inf* such that, for each S, S^{fin} yields the finite and S^{inf} the infinite part of the execution of S. In the course of proving that these mappings have the desired properties, we discovered a rather general proof technique for showing properties of recursive procedures which can be applied to a variety of problems not necessarily related to that of infinite computations.

An important source of inspiration for our paper was provided by Nivat's investigations of infinite words generated by context free grammars (e.g. [2,6,12,13,14]). In an *operational* semantics, execution of a statement S may be seen as the generation of a sequence of elementary actions, and an infinite execution then corresponds to an infinite word in the language of all possible execution sequences corresponding to the (nondeterministic) statement S. In our paper we do not make these operational notions precise, but stick to the denotational approach. Though the way the problems appear here is at first sight quite different, there is a surprisingly close structural resemblance between the results of language theory and of denotational semantics. More specifically, the definitions of *fin* and *inf* for the *regular* case (statements with only iteration, no full recursion) are of exactly the same form as certain results in Nivat's work (mentioned e.g. in [6]), and the definition of the general case (statements with full recursion) is - after some appropriate transliteration - strikingly similar to theorem 1 of NIVAT [13]. A new element in our considerations is that through the semantic approach we obtain a better understanding of the underlying structure of these results. We shall show that they ultimately rely on a certain simple - and purely semantic - property of fixed points. We thus hope to clarify the problem which at first may seem purely syntactic in nature in that it concerns manipulations with program texts or with infinite derivations in language theory. In fact, the fixed point property referred to here appears to be at the heart of a number of seemingly unrelated problems concerning, e.g., properties of weakest preconditions studied in Chapter 8 of DE BAKKER [4]. Briefly, the following argument may be applied for each of these questions: Suppose we want to justify a certain syntactic mapping which is intended to embody a certain semantic feature. Normally, such a justification proceeds by an inductive proof on the syntactic structure of the statements involved. Now a central result of our paper is that, provided a number of rather general conditions are fulfilled, it is only necessary to check those cases of the induction which are not concerned with the iteration or recursion constructs. Only the, say, straight-line cases have to be considered individually, and the iteration or recursion cases are obtained as it were for free from a general definitional scheme.

Our paper is organized in six sections. You are now reading Section 1 which gives the introduction. In Section 2 we define syntax and semantics of the two languages we consider, one with only iteration (essentially as provided by the while statement or the <u>do</u>-<u>od</u> guarded command), and the other with full recursion in the form of parameterless recursive procedures. We consider these in the syntactic form of the μ-calculus ([5,7]), since this is a convenient tool for the mathematical analysis we have in mind. In Section 5 we translate our results to a more traditional framework with declarations of mutually recursive (parameterless) procedures. A secondary feature of our language is a systematic treatment of the notions of *failure* and *abortion*. Contrary to the approach taken by other authors (such as [1]), we include the empty set (of states) in our considerations and use it to model failure of a statement. In this way, failure leaves no trace in the output. Abortion, on the other hand, does leave a trace behind in the form of a special abort state (for which we use δ). Our way of treating failure has, we think, advantages in that it allows us to express a variety of constructs involving tests (such as the conditional statement, while statement and guarded commands) all using just one "test statement" in our language. As a side remark we add here that the empty set can conveniently be used to model *waiting* in a context with concurrency, whereas an abort outcome should be used in case a deadlock situation occurs which one wants to be signalled. Apart from the introduction of the abort construct, the definitions of Section 2 follow closely those of Chapter 7 of [4]. In Section 3 we give a simple version of our main result, viz. for the case of regular statements (with only iteration). The general case follows in Section 4. Here the fixed point lemma mentioned above is proved, and it is shown how - in a rather general setting - the relationship between syntactic and semantic mappings between (meanings of) statements can be analyzed such that the recursion case is obtained as it were automatically. This part of the paper is rather abstract, and we provide some concrete applications of the techniques in the subsequent sections. In Section 5 we reformulate our result for systems of recursive procedures - rather than for statements in the μ-calculus -, and clarify its close structural similarity to Nivat's theorem. In Section 6 we study a variety of weakest preconditions (to be compared to a similar variety in an operational framework as investigated by HAREL [8]), and obtain certain fixed point results for the regular case by straightforward application of the general strategy of Section 4 - rather than, as in Chapter 8 of [4], by using more or less elaborate arguments in each specific case. Finally, we briefly mention some further applications which obviate some of the complications in the proofs of [4].

The first author gratefully acknowledges the hospitality of Bar Ilan University and the Weizmann Institute during July 1981. The members of the MC Working Group on Semantics formed the first audience for the ideas presented here.

2. SYNTAX AND SEMANTICS

We shall be concerned with two simple languages, one with only iteration and the other with full recursion. The former is actually a special case of the latter, and introduced primarily for didactic reasons. Both languages contain simple integer and boolean expressions, together with assignment, composition and nondeterministic choice. The way boolean expressions are used as statements is somewhat unusual, and will be explained later in the section. A special symbol Δ is introduced for the *abort* statement.

The following notations are used for the respective syntactic classes (here and below we use the convention that the phrase "(m $\in$)M such that ..." introduces a set M, with typical elements m ranging over M, such that ...):

(n $\in$) *Icon*: *integer constants*
(x $\in$) *Ivar*: *integer variables*
(s $\in$) *Iexp*: *integer expressions*
(b $\in$) *Bexp*: *boolean expressions*
(R $\in$) *Regs*: *regular statements*
(S $\in$) *Stat*: (general) *statements*
(X $\in$) *Stmv*: *statement variables*
(serving the same role as procedure variables P in a more orthodox syntax).

The classes *Ivar* and *Stmv* are arbitrary disjoint infinite sets of symbols - assumed well-ordered for technical convenience. The structure of the elements of *Icon* is left unspecified. The other classes are defined using a self-explanatory variant of the Backus-Naur formalism in

DEFINITION 2.1 (syntax).

a. (integer expressions)

$$s ::= n \mid x \mid s_1 + s_2 \mid \ldots \mid \underline{\text{if}}\ b\ \underline{\text{then}}\ s_1\ \underline{\text{else}}\ s_2\ \underline{\text{fi}}$$

b. (boolean expressions)

$$b ::= \underline{\text{true}} \mid \underline{\text{false}} \mid s_1 = s_2 \mid \ldots \mid \neg b \mid b_1 \supset b_2$$

c. (regular statements)

$R::= \quad x:=s\,|\,b\,|\,\Delta\,|\,R_1;R_2\,|\,R_1 \cup R_2\,|\,R^\dagger$

d. (general statements)

$S::= \quad x:=s\,|\,b\,|\,\Delta\,|\,S_1;S_2\,|\,S_1 \cup S_2\,|\,X\,|\,\mu X[S].$

Remarks

1. At the place of the ... in clauses a and b, other operators (−,≤,...) can be added. In fact, we could omit all specialization to the domain of integers, and introduce arbitrary function and relation symbols in our expressions. All results to be obtained below hold for (interpretations over) arbitrary structures, and we stick to the integers only for ease of presentation.
2. Boolean expressions as statements may appear somewhat unusual. They were introduced as such in [5], and reappear, e.g., in dynamic logic [8] as test statements (p?). In the framework of denotational semantics - to be introduced in a moment - a statement determines a mapping from states to sets of states. A boolean b - viewed as a statement - maps a state either to itself (for b true in that state) or to the empty set of states (for b false in that state). In the latter case, b may be said to *fail*. This is a special case of a property of statements S in general, viz. the possibility of their failure which is modelled by delivery of the empty set. Failure should be contrasted with abortion, appearing in our system through the atomic statement Δ which aborts for all input states. Abortion is modelled by delivering a special abort state δ as output, whereas nontermination is reflected in the usual way by yielding the undefined or bottom state ⊥.
3. " ∪ " denotes nondeterministic choice: Executing $R_1 \cup R_2$ or $S_1 \cup S_2$ means executing R_1 or R_2 (S_1 or S_2).
4. $R^\dagger$ denotes finite *or infinite* repetition of the statement R. It should be contrasted with the construct R^* which is often used in similar investigations, usually referring only to arbitrary *finite* repetition of R. (In a purely relational theory, the difference between R^* and $R^\dagger$ remains unobserved since an infinite computation always yields an empty output set.) Using R^ω for infinite repetition of R, we have that $R^\dagger$ is equivalent to $R^* \cup R^\omega$. (We prefer " † " - used in the theory of infinite

words by, e.g., PARK [15] - to "∞" - as used e.g. by NIVAT [2,6,12,13,14].)

5. $\mu X[S]$ is a construct taken from the μ-calculus ([5,9]), denoting a call of a parameterless recursive procedure. The prefix μX in $\mu X[S]$ *binds* occurrences of X in S, and, for S of the form ...X...X..., executing $\mu X[S]$ corresponds to a call - in a language with a more familiar syntax - of a procedure P declared by $P \Leftarrow \ldots P \ldots P \ldots$. In case of a system of, say, two declarations $P_1 \Leftarrow S_1(P_1,P_2)$, $P_2 \Leftarrow S_2(P_1,P_2)$ ((...) denoting possible free occurrences of ..., *not* application), the construct in the μ-calculus corresponding to a call of P_1 is $\mu X_1[S_1(X_1,\mu X_2[S_2(X_1,X_2)])]$. Much more about this can be found in [4]. A statement S without free occurrences of statement variables is called *closed*.

We use "$\equiv$" for syntactic identity, and substitution of S' for X in S - applying the usual renaming of bound statement variables to prevent clashes - is denoted by $S[S'/X]$.

In order to help the reader's understanding of our syntax we now list a number of constructs in the syntax of an ALGOL-like or guarded command language ([7]), and then present the corresponding construct in our language(s):

$\underline{\text{if}}\ b\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}} \rightsquigarrow (b;S_1) \cup (\neg b;S_2)$

$\underline{\text{while}}\ b\ \underline{\text{do}}\ R\ \underline{\text{od}} \rightsquigarrow (b;R)^{\dagger};\neg b$

$\underline{\text{if}}\ b_1 \to R_1 \square \ldots \square b_n \to R_n\ \underline{\text{fi}} \rightsquigarrow (b_1;R_1) \cup \ldots \cup (b_n;R_n) \cup (\neg b_1 \wedge \ldots \wedge \neg b_n;\Delta)$

$\underline{\text{do}}\ b_1 \to R_1 \square \ldots \square b_n \to R_n\ \underline{\text{od}} \rightsquigarrow \underline{\text{while}}\ b_1 \vee \ldots \vee b_n\ \underline{\text{do}}\ (b_1;R_1) \cup \ldots \cup (b_n;R_n)\ \underline{\text{od}}$

$\left.\begin{array}{l}\underline{\text{fail}} \rightsquigarrow \underline{\text{false}} \\ \underline{\text{skip}} \rightsquigarrow \underline{\text{true}}\end{array}\right\}$ note that these boolean expressions are indeed statements

$\underline{\text{abort}} \rightsquigarrow \Delta$

$\underline{\text{while}}\ b\ \underline{\text{do}}\ S\ \underline{\text{od}} \rightsquigarrow \mu X[(b;S;X) \cup \neg b]$ (X not free in S)

(These correspondences work well in a sequential context. In the presence of concurrency, complications may arise. We know how to deal with these, but leave an explanation of such issues to a future paper.)

This concludes our discussion of the syntactic aspects of our languages, and we next turn to their semantics. We begin with a quick introduction to the theory of complete partially ordered sets (cpo's). For details and proofs we refer to, e.g., [4]. A cpo's a pair $(C,\sqsubseteq)$ with C a non-empty set and "$\sqsubseteq$" a partial order on C, such that (i) there is a *least* element $\perp_C$ with $\perp_C \sqsubseteq x$ for all $x \in C$, and (ii) each ascending $\sqsubseteq$-chain $\langle x_i \rangle_i$ has a least upper bound $\bigsqcup_i x_i$. Usually, explicit mentioning of the ordering "$\sqsubseteq$" in a cpo$(C,\sqsubseteq)$ is omitted; similarly for the index C in $\perp_C$. For cpo's C_1, C_2, $C_1 \times C_2$ is defined as a cpo in the natural way through component-wise ordering. We call $f: C_1 \to C_2$ *strict* whenever $f(\perp) = \perp$, and *monotonic* whenever if $x_1 \sqsubseteq x_2$ then $f(x_1) \sqsubseteq f(x_2)$. The class of all strict (monotonic) functions $C_1 \to C_2$ is denoted by $C_1 \to_s C_2$ $(C_1 \to_m C_2)$. A monotonic function f is called *continuous* whenever, for each chain $\langle x_i \rangle_i$ in C_1, we have $f(\bigsqcup_i x_i) = \bigsqcup_i f(x_i)$. For $f,g: C_1 \to C_2$, we put $f \sqsubseteq g$ whenever $f(x) \sqsubseteq g(x)$ for all $x \in C_1$. Two important properties of cpo's are: (i) For C_1, C_2 cpo's, the class of all continuous functions $C_1 \to C_2$ (denoted by $[C_1 \to C_2]$) is a cpo, and (ii) Each continuous $f: C \to C$ has a least fixed point (ℓfp) μf (i.e., $f(\mu f) = \mu f$, and $f(y) \sqsubseteq y \Rightarrow \mu f \sqsubseteq y$) obtained as $\mu f = \bigsqcup_i f^i(\perp)$ (where $f^i = f \circ f \circ \ldots \circ f$, i factors f). Often, we shall encounter *flat* cpo's: C is called flat whenever, for all $x_1, x_2 \in C$, $x_1 \sqsubseteq x_2$ iff $x_1 = \perp$ or $x_1 = x_2$. Occasionally we shall need the following further definitions: A cpo C is a *complete lattice* whenever each subset $X \sqsubseteq C$ has a least upper bound $\bigsqcup X$ and (hence) a greatest lower bound $\sqcap X$. For C a complete lattice and $f: C \to_m C$, the least fixed point μf and greatest fixed point νf of f exist. We call $f: C_1 \to C_2$ *antimonotonic* whenever if $x_1 \sqsubseteq x_2$ then $f(x_2) \sqsubseteq f(x_1)$, and an antimonotonic $f: C_1 \to C_2$ is called *anticontinuous* (for C_2, e.g., a complete lattice) whenever for each ascending $\sqsubseteq$-chain $\langle x_i \rangle_i$ we have $f(\bigsqcup_i x_i) = \sqcap_i f(x_i)$.

Throughout the paper we use the λ-notation for functions: For example, $\lambda x.x$ denotes the identity function: $C \to C$, and for $f \in [C_1 \times C_2 \to C_2]$, $\mu[\lambda y.f(x,y)]$ $(\in C_2)$ is the least fixed point of the function $\lambda y.f(x,y)$ in $[C_2 \to C_2]$.

Next, we introduce the semantic notion of *state*. Let $(\sigma \in)$ Σ denote the set of all states. We define $\Sigma = \Sigma_0 \cup \{\delta\} \cup \{\perp\}$, where Σ_0 is the set of *proper states*, $\Sigma_0 = Ivar \to \mathbb{Z}$ ($\mathbb{Z}$ the set of integers). Moreover, δ is a special state (the abort state) with $\delta \notin \Sigma_0$, and $\perp$ is a special state $(\notin \Sigma_0 \cup \{\delta\})$, the bottom state. We turn Σ into a flat cpo by putting, for each $\sigma_1, \sigma_2 \in \Sigma$, $\sigma_1 \sqsubseteq \sigma_2$ iff $\sigma_1 = \perp$ or $\sigma_1 = \sigma_2$. Let $\mathbb{Z}_\perp = \mathbb{Z} \cup \{\perp_{\mathbb{Z}}\}$, $W_\perp = W \cup \{\perp_W\}$, where $W = \{tt,ff\}$ is the set of truth-values. $\mathbb{Z}_\perp$ and $W_\perp$ are taken as flat cpo's. Let, moreover, for $\sigma \in \Sigma_0$ and $\alpha \in \mathbb{Z}$, $\sigma\{\alpha/x\}$ denote the proper state such that $\sigma\{\alpha/x_1\}(x_2) = \alpha$ for $x_1 \equiv x_2$, and $\sigma\{\alpha/x_1\}(x_2) = \sigma(x_2)$ for $x_1 \not\equiv x_2$.

For a language with nondeterminacy, the meaning of a statement is a mapping from states to sets of states. For the languages dealt with in our paper it is sufficient to consider only the collection T of all those subsets of Σ which, when infinite, contain $\perp$. (This is a consequence of the fact that our languages are of *bounded* nondeterminacy. In an operational semantics, the computation tree modelling execution for a given input state is finitely branching and therefore it allows application of König's lemma. An infinite path in the tree is, denotationally, reflected by the presence of $\perp$ in the output set, and whenever the output set (set of states labelling the leaves of the tree) is infinite, $\perp$ has to be in the set. We shall not say more about this here; the reader may consult [1,3,4,7,10] for more information.) On the elements $\tau \in T$ the so-called Egli-Milner ordering is defined:

DEFINITION 2.2.

a. $\tau_1 \sqsubseteq \tau_2$ iff either $\perp \in \tau_1$ and $\tau_1 \backslash \{\perp\} \subseteq \tau_2$ ($\subseteq$ is set inclusion) or $\perp \notin \tau_1$ and $\tau_1 = \tau_2$.

b. Let, for $\psi\colon \Sigma \to_s T$, $\hat{\psi}\colon T \to_s T$ be defined by $\hat{\psi} = \lambda\tau. \bigcup_{\sigma\in\tau} \psi(\sigma)$ let $\psi_1 \circ \psi_2 = \lambda\sigma.\hat{\psi}_1(\psi_2(\sigma))$ and $\psi_2 \cup \psi_2 = \lambda\sigma.\psi_1(\sigma) \cup \psi_2(\sigma)$.

c. $M \overset{\text{df.}}{=} [\Sigma \to_s T]$, and ϕ denotes a typical element of M.

A justification of this definition is contained in

LEMMA 2.3.

a. $(T,\sqsubseteq)$ is a cpo, where, for a chain $<\tau_i>_i = \begin{cases} \cup\tau_i, & \text{if } \perp \in \tau_i \text{ for all } i \\ \tau_{i_0}, & \text{if } \perp \notin \tau_{i_0} \text{ (for some } i_0) \end{cases}$ (where "$\cup$" denotes set-theoretic union)

b. "$\hat{\ }$" is a continuous mapping: $(\Sigma \to_s T) \to (T \to_s T)$, and, for ϕ continuous $\hat{\phi}$ is continuous.

c. Both "o" and "$\cup$" are continuous in both their arguments.

PROOF. See, e.g., [4].

Remark. We observe that $\emptyset$ and $\{\delta\}$ are maximal elements of T in the Egli-Milner ordering. This mirrors the fact that a statement which fails or aborts cannot be extended to a statement containing more information. On the other hand, $\{\bot\} \sqsubseteq \tau$ holds for all τ, and, in particular, $\{\bot\} \sqsubseteq \emptyset$ holds; hence, $\emptyset$ is not the least element of T.

In the non-regular case we need, besides states assigning meaning to integer variables, also *environments* assigning meaning to statement variables. We take $(\varepsilon\in)\ E \overset{\text{df.}}{=} \mathit{Stmv} \to M$, and use the notation $\varepsilon\{\phi/X\}$ analogous to the $\sigma\{\alpha/x\}$ notation.

We now introduce the valuation functions V, W, R and M, of the following types:

$$V\colon \mathit{Iexp} \to (\Sigma \to \mathbb{Z}_\bot)$$
$$W\colon \mathit{Bexp} \to (\Sigma \to W_\bot)$$
$$R\colon \mathit{Regs} \to M$$
$$M\colon \mathit{Stat} \to (E \to M)$$

Their definitions are given in

<u>DEFINITION 2.4</u> (semantics).

a. $V(s)(\delta) = V(s)(\bot) = \bot_{\mathbb{Z}}$, and for $\sigma \neq \delta,\bot$, $V(s)(\sigma)$ has the usual meaning (e.g., $V(x)(\sigma) = \sigma(x)$, etc.; for details see [4]).

b. $W(b)(\delta) = W(b)(\bot) = \bot_W$, and, for $\sigma \neq \delta,\bot$, $W(b)(\sigma)$ has the usual meaning (e.g., $W(s_1{=}s_2)(\sigma) = (V(s_1)(\sigma) = V(s_2)(\sigma))$, etc.).

c. $R(R)(\sigma) = \{\sigma\}$ if $\sigma = \delta$ or $\sigma = \bot$, and, for $\sigma \neq \delta$, (by convention, $\lambda\sigma....$ is short for $\lambda\sigma\in\Sigma....$):

$$R(x{:=}s) = \lambda\sigma\{\sigma\{\,V(s)(\sigma)/x\}\}$$
$$R(b) = \lambda\sigma.\ \underline{\text{if}}\ W(b)(\sigma)\ \underline{\text{then}}\ \{\sigma\}\ \underline{\text{else}}\ \emptyset\ \underline{\text{fi}}$$
$$R(\Delta) = \lambda\sigma.\{\delta\}$$
$$R(R_1;R_2) = R(R_2)\circ R(R_1)$$
$$R(R_1 \cup R_2) = R(R_1) \cup R(R_2)$$
$$R(R^\dagger) = \bigsqcup_i \phi_i, \text{ where}$$

$$\phi_0 = \lambda\sigma.\{\bot\}$$
$$\phi_{i+1} = (\phi_i \circ R(R)) \cup (\lambda\sigma.\{\sigma\})$$

d. $M(S)(\varepsilon)(\sigma) = \{\sigma\}$ if $\sigma = \delta$ or $\sigma = \bot$, and, for $\sigma \neq \delta,\bot$,

$M(x{:=}s)(\varepsilon) = \lambda\sigma.\{\sigma\{V(s)(\sigma)/x\}\},\ldots,M(S_1 \cup S_2)(\varepsilon) = M(S_1)(\varepsilon) \cup M(S_2)(\varepsilon)$

$M(X)(\varepsilon) = \varepsilon(X)$,
$M(\mu X[S])(\varepsilon) = \mu[\lambda\phi.M(S)(\varepsilon\{\phi/X\})]$.

Remarks

1. The mapping $\Phi = \lambda\phi.M(S)(\varepsilon\{\phi/X\})$ in clause d is continuous (i.e., $\Phi \in [M \to M]$) and, therefore, has a least fixed point $\mu\Phi$.
2. Let us assume - for the purpose of our theory rather than as language extensions for their own sake - that the syntax of *Regs* is extended with

$$R ::= \ldots | R^* | R^\omega.$$

As definition of their semantics we give:
$R(R^*) = \bigcup_i \psi_i$ (lub with respect to set-inclusion), where

$$\psi_0 = \lambda\sigma.\emptyset$$
$$\psi_{i+1} = (\psi_i \circ R(R)) \cup (\lambda\sigma.\{\sigma\})$$

and

$$R(R^\omega) = \bigsqcup_i \chi_i, \text{ where}$$
$$\chi_0 = \lambda\sigma.\{\bot\}$$
$$\chi_{i+1} = \chi_i \circ R(R).$$

We leave to the reader the proof that, indeed, $R(R^\dagger) = R(R^*) \cup R(R^\omega)$. Another way of viewing the difference between $R^\dagger$ and R^* is the following: Let Ω denote the statement that terminates nowhere (i.e., $R(\Omega) = \lambda\sigma.$ <u>if</u> $\sigma \neq \delta$ <u>then</u> $\{\bot\}$ <u>else</u> $\{\delta\}$ <u>fi</u>, and let $R_1 \sqsubseteq R_2$ abbreviate $R(R_1)(\sigma) \sqsubseteq R(R_2)(\sigma)$ for all σ, and similarly for $R_1 \subseteq R_2$. We now have that - using an informal terminology - $R^\dagger$ corresponds to the least upperbound of the $\sqsubseteq$-chain

$$\Omega \sqsubseteq (R;\Omega) \cup \underline{\text{true}} \sqsubseteq \ldots \sqsubseteq (R^i;\Omega) \cup R^{i-1} \cup \ldots \cup R \cup \underline{\text{true}} \sqsubseteq \ldots$$

(where R^i stands for $R;\ldots;R$ (i times), and the equivalence R; <u>true</u> = R is used), and R^* is the least upperbound of the $\subseteq$-chain

$$\underline{\text{false}} \subseteq R; \underline{\text{false}} \cup \underline{\text{true}} \subseteq \ldots \subseteq R^{i-1} \cup \ldots \cup R \cup \underline{\text{true}} \subseteq \ldots$$

Here we have used that, for all R, R; false = false. Note that $R;\Omega = \Omega$ *only* holds when R fails nowhere. This is a consequence of the fact that $\hat{\phi}(\emptyset) = \emptyset$ holds for all ϕ; in particular, $\mathcal{R}(\Omega)^{\wedge}(\emptyset) = \emptyset$.

3. In section 4 we shall introduce a construct in an extension of *Stat* which plays the same role with respect to $\mu X[S]$ as R^* plays with respect to $R^\dagger$.

3. INFINITE COMPUTATIONS: THE REGULAR CASE

For each regular R, we syntactically define constructs R^{fin} and R^{inf} where R^{fin} (R^{inf}) denotes that part of R which gives precisely the finite (infinite) part of the computation. The general problem (for any $S \in$ *Stat*) is addressed in the next section; in the present one we only deal with the regular case. No proofs are given since the results are just specializations of the general case.

DEFINITION 3.1 (semantic finite and infinite parts).

a. For $\tau \in T$, we put $\tau^{fin} = \tau\backslash\{\bot\}$, $\tau^{inf} = \tau\backslash\tau^{fin}$ (where "\" denotes set-theoretic difference).

b. For $\phi \in M$, we put $\phi^{fin} = \lambda\sigma.\phi(\sigma)^{fin}$ and $\phi^{inf} = \lambda\sigma.\phi(\sigma)^{inf}$.

We can now give a precise formulation of the aim of this section: For $R \in$ *Regs*, define syntactically constructs R^{fin} and R^{inf} such that $\mathcal{R}(R^{fin}) = \mathcal{R}(R)^{fin}$, $\mathcal{R}(R^{inf}) = \mathcal{R}(R)^{inf}$. From now on, we assume syntax and semantics of *Regs* extended as described in remark 2 after definition 2.4. The following definition gives the desired construction:

DEFINITION 3.2 (syntactic finite and infinite parts).

a.
$$\begin{aligned}
(x{:=}s)^{fin} &\equiv x{:=}s \\
b^{fin} &\equiv b \\
\Delta^{fin} &\equiv \Delta \\
(R_1;R_2)^{fin} &\equiv R_1^{fin};\ R_2^{fin} \\
(R_1 \cup R_2)^{fin} &\equiv R_1^{fin} \cup R_2^{fin} \\
R^{\dagger fin} &\equiv R^{fin*}
\end{aligned}$$

b. $(x:=s)^{inf} \equiv \underline{\text{false}}$
$b^{inf} \equiv \underline{\text{false}}$
$(R_1;R_2)^{inf} \equiv R_1^{inf} \cup R_1^{fin}; R_2^{inf}$
$(R_1 \cup R_2)^{inf} \equiv R_1^{inf} \cup R_2^{inf}$
$R^{\dagger inf} \equiv R^{fin*}; R^{inf} \cup R^{fin\omega}$.

Remarks

1. Not surprisingly, these formulae have exactly the same structure as the formulae appearing in the theory of languages with infinite words (e.g. [6]). In fact, the primary motivation for the present research was our wish to study these formulae in the framework of denotational semantics, together with their generalization for the non-regular case, and to investigate the foundations of the proof of their justification.
2. Though we do not really need them, for completeness sake are also give the formulae for R^* and R^ω:

$$R^{*fin} \equiv R^{fin*} \qquad R^{*inf} \equiv R^{fin*};R^{inf}$$

$$R^{\omega fin} \equiv \underline{\text{false}} \qquad R^{\omega inf} \equiv R^{fin*}; R^{inf} \cup R^{fin\omega}.$$

3. Some understanding for the structure of the formulae for $R^{\dagger inf}$ can be obtained by using the fact that $R^\dagger = R^* \cup R^\omega = \underline{\text{true}} \cup R \cup R;R \cup \ldots \cup R^k \cup \ldots \cup R^\omega$, and the formulae for $(R_1 \cup R_2)^{inf}$ and $(R_1;R_2)^{inf}$. We have

$$\begin{aligned}
R^{\dagger inf} &= (\underline{\text{true}} \cup R \cup R^2 \cup \ldots \cup R^k \cup \ldots \cup R^\omega)^{inf} \\
&= \underline{\text{true}}^{inf} \cup R^{inf} \cup (R^2)^{inf} \cup \ldots \cup (R^k)^{inf} \cup \ldots \cup (R^\omega)^{inf} \\
&= \underline{\text{false}} \cup R^{inf} \cup (R^{inf} \cup R^{fin};R^{inf}) \cup \ldots \\
&\quad \cup (R^{inf} \cup R^{fin};(R^{k-1})^{inf}) \cup \ldots \cup (R^{inf} \cup R^{fin};(R^\omega)^{inf}) \\
&= (\text{after } \omega \text{ iterations}) \\
&\quad (\underline{\text{true}} \cup R^{fin} \cup \ldots \cup (R^{fin})^k \cup \ldots);R^{inf} \cup (R^{fin})^\omega \\
&= R^{fin*};R^{inf} \cup (R^{fin})^\omega
\end{aligned}$$

(Note that we do not claim this to be a proof of anything.)

The next theorem expresses the desired result:

THEOREM 3.3. *For each* R $\in$ *Regs*,

a. $\mathcal{R}(R^{fin}) = \mathcal{R}(R)^{fin}$

   $\mathcal{R}(R^{inf}) = \mathcal{R}(R)^{inf}$

b. $\mathcal{R}(R) = \mathcal{R}(R^{fin}) \cup \mathcal{R}(R^{inf})$

PROOF.

a. Special case of theorem 4.7.

b. Immediate from part a and the fact that $\mathcal{R}(R) = \mathcal{R}(R)^{fin} \cup \mathcal{R}(R)^{inf}$ (since $\tau = \tau^{fin} \cup \tau^{inf}$).

4. INFINITE COMPUTATIONS: THE GENERAL CASE

This section presents our treatment of infinite computations in the general case. We first introduce some auxiliary syntactic (and associated semantic) definitions. Next, we give the definitions of S^{fin} and S^{inf}. Their justification is based on (i) a general (semantic) lemma on properties of fixed points (lemma 4.3), and (ii) a - generally applicable - theorem enabling us to connect syntactic transformations with semantic ones (theorem 4.5). Once theorem 4.5 has been established, it is straightforward to prove that the definitions of *fin* and *inf* are indeed the desired ones.

The auxiliary syntactic construct we introduce plays the same role with respect to $\mu X[S]$ as R^* plays with respect to $R^\dagger$.

DEFINITION 4.1 (auxiliary and extended statements).

a. Let (A$\in$) *Auxs* be the class of *auxiliary statements*. Let (Y$\in$) *Auxv* be the class of *auxiliary statement variables*. We define

$$A ::= x:=s \mid b \mid \Delta \mid A_1 ; A_2 \mid A_1 \cup A_2 \mid Y \mid \alpha Y[A]$$

   (see remark 1)

b. Let (T$\in$) *Exts* be the class of *extended statements*. (There is no need to introduce a separate class of extended statement variables (X$\in$) *Stmv* serves our purpose here.)

$$T ::= x:=s \mid b \mid \Delta \mid T_1;T_2 \mid T_1 \cup T_2 \mid X \mid \mu X[T] \mid A$$

c. Let $(M,\sqsubseteq) = [\Sigma \to_s T]$ be as before. Let $(M,\subseteq)$ be the cpo of continuous functions $\psi\colon \Sigma \to_s T$ ordered by set-inclusion (i.e. $\psi_1 \subseteq \psi_2$ iff $\psi_1(\sigma) \subseteq \psi_2(\sigma)$ for all σ; recall that $\psi_1(\sigma)$, $\psi_2(\sigma)$ are *sets* in T.) For $\Phi \in [(M,\sqsubseteq) \to (M,\sqsubseteq)]$, $\mu_{\sqsubseteq}\Phi$ denotes its least fixed point with respect to " $\sqsubseteq$ ", and for $\Psi \in [(M,\subseteq) \to (M,\subseteq)]$, $\mu_{\subseteq}\Psi$ denotes its least fixed point with respect to "$\subseteq$". The class of environments E is extended to mappings $(\mathit{Stmv} \cup \mathit{Auxv}) \to M$. We define the valuations $\mathcal{A}\colon \mathit{Auxs} \to (E \to M)$, $\mathcal{T}\colon \mathit{Exts} \to (E \to M)$ as follows:

$\mathcal{A}(A)(\varepsilon)(\sigma) = \{\sigma\}$ for $\sigma = \delta$ or $\sigma = \bot$, and similarly for $\mathcal{T}(T)(\varepsilon)(\sigma)$.

Otherwise,

$\mathcal{A}(x:=s)(\varepsilon) = \lambda\sigma.\{\sigma\{\mathcal{V}(s)(\sigma)/x\}\}, \ldots, \mathcal{A}(A_1 \cup A_2)(\varepsilon) = \mathcal{A}(A_1)(\varepsilon) \cup \mathcal{A}(A_2)(\varepsilon)$,

$\mathcal{T}(x:=s)(\varepsilon) = \lambda\sigma.\{\sigma\{\mathcal{V}(s)(\sigma)/x\}\}, \ldots, \mathcal{T}(T_1 \cup T_2)(\varepsilon) = \mathcal{T}(T_1)(\varepsilon) \cup \mathcal{T}(T_2)(\varepsilon)$

$\mathcal{A}(Y)(\varepsilon) = \varepsilon(Y), \mathcal{T}(X)(\varepsilon) = \varepsilon(X)$

$\mathcal{A}(\alpha Y[A])(\varepsilon) = \mu_{\subseteq}[\lambda\psi.\mathcal{A}(A)(\varepsilon\{\psi/Y\})]$

$\mathcal{T}(\mu X[T])(\varepsilon) = \mu_{\sqsubseteq}[\lambda\phi.\mathcal{T}(T)(\varepsilon\{\phi/X\})]$

$\mathcal{T}(A)(\varepsilon) = \mathcal{A}(A)(\bar{\varepsilon})$

Remarks

1. Auxiliary statements $A \in \mathit{Auxs}$ are syntactically isomorphic to statements $S \in \mathit{Stat}$. The only difference is in their semantics in that in defining the meaning of the $\alpha Y[A]$ construct we use least fixed points with respect to the $\subseteq$-ordering. (To emphasize the difference we use a different notation (α rather than μ) for recursive constructs.)
2. Extended statements combine the structure of ordinary (S-type) and auxiliary (A-type) statements. In particular, $\mathit{Stat} \subseteq \mathit{Exts}$ and $\mathit{Auxs} \subseteq \mathit{Exts}$. Note, however, that nested applications of recursive constructs of the form $\mu X[\ldots\alpha Y[A]\ldots]$ or $\alpha Y[\ldots\mu X[T]\ldots]$ with X free in A or Y free in T are not included. As a consequence, no complications are encountered in the verification of the usual continuity properties of $\Psi = \lambda\psi.\mathcal{A}(A)(\varepsilon\{\psi/Y\})$, for which $\Psi \in [(M,\subseteq) \to (M,\subseteq)]$ holds, or of $\Phi = \lambda\phi.\mathcal{T}(T)(\varepsilon\{\phi/X\})$, for which $\Phi \in [(M,\sqsubseteq) \to (M,\sqsubseteq)]$ holds.
3. For subsequent use, we observe that it is straightforward to verify that

$$\mathcal{M}(S[S'/X])(\varepsilon) = \mathcal{M}(S)(\varepsilon\{\mathcal{M}(S')(\varepsilon)/X\}) \tag{4.1a}$$

(4.1b) $\mathcal{A}(A[A'/Y])(\varepsilon) = \mathcal{A}(A)(\varepsilon\{\mathcal{A}(A')(\varepsilon)/Y\})$

(4.1c) $\mathcal{T}(T[T'/X])(\varepsilon) = \mathcal{T}(T)(\varepsilon\{\mathcal{T}(T')(\varepsilon)/X\})$.

4. Note that $\mu X[S]$ can be viewed - again using an informal terminology - as least upper bound of the $\sqsubseteq$-chain

$$\Omega \sqsubseteq S[\Omega/X] \sqsubseteq S[S[\Omega/X]/X] \sqsubseteq \ldots$$

whereas $\alpha Y[A]$ is least upper bound of the $\subseteq$-chain

$$\underline{\text{false}} \subseteq S[\underline{\text{false}}/Y] \subseteq S[S[\underline{\text{false}}/Y]/Y].$$

5. The way in which the regular statements can be embedded in the class of general or extended statements is given by the following correspondence:

$$R^{\dagger} \rightsquigarrow \mu X[R;X \cup \underline{\text{true}}]$$
$$R^{*} \rightsquigarrow \alpha Y[R;Y \cup \underline{\text{true}}]$$
$$R^{\omega} \rightsquigarrow \mu X[R;X]$$

(Remember that R has, by its definition, no free occurrences of X or Y.) Two further correspondences we shall have occasion to use, are

$$\mu X[R_1;X \cup R_2] \rightsquigarrow R_1^{*};R_2 \cup R_1^{\omega}$$
$$\alpha Y[R_1;Y \cup R_2] \rightsquigarrow R_1^{*};R_2.$$

We now arrive at the central definition of our paper, viz. of S^{fin} and S^{inf}. Let, for each $X \in \mathit{Stmv}$, X^{fin} be some element in Auxv and X^{inf} an element in Stmv. We assume, moreover, that $X_1 \not\equiv X_2 \Rightarrow X_1^{fin} \not\equiv X_2^{fin}$, $X_1^{inf} \not\equiv X_2^{inf}$. For arbitrary S, we define $S^{fin} \in \mathit{Auxs}$ and $S^{inf} \in \mathit{Exts}$ by

DEFINITION 4.2 (syntactic *fin* and *inf*).

a. $(x:=s)^{fin} \equiv x:=s$

$b^{fin} \equiv b$

$\Delta^{fin} \equiv \Delta$

$$
\begin{array}{lll}
& (S_1;S_2)^{fin} & \equiv S_1^{fin};S_2^{fin} \\
& (S_1 \cup S_2)^{fin} & \equiv S_1^{fin} \cup S_2^{fin} \\
& \mu X[S]^{fin} & \equiv \alpha X^{fin}[S^{fin}] \\
\text{b.} & (x:=s)^{inf} & \equiv \underline{\text{false}} \\
& b^{inf} & \equiv \underline{\text{false}} \\
& \Delta^{inf} & \equiv \underline{\text{false}} \\
& (S_1;S_2)^{inf} & \equiv S_1^{inf} \cup S_1^{fin};S_2^{inf} \\
& (S_1 \cup S_2)^{inf} & \equiv S_1^{inf} \cup S_2^{inf} \\
& \mu X[S]^{inf} & \equiv \mu X^{inf}[S^{inf}[\mu X[S]^{fin}/X^{fin}]]
\end{array}
$$

Remarks

1. We leave it to the reader to verify that, indeed, $S^{fin} \in Auxs$, $S^{inf} \in Exts$.
2. Apart from the definitions for the μ-construct, the definitions are exactly as in definition 3.2.

By way of example, we show how the formulae of definition 3.2 can be obtained as special cases of definition 4.2. Let R be any regular statement.

$$
\begin{aligned}
R^{\dagger fin} &\sim\!\sim \mu X[R;X \cup \underline{\text{true}}]^{fin} \sim\!\sim \alpha X^{fin}[(R;X \cup \underline{\text{true}})^{fin}] \\
&\sim\!\sim \alpha X^{fin}[R^{fin};X^{fin} \cup \underline{\text{true}}^{fin}] \\
&\sim\!\sim \alpha X^{fin}[R^{fin};X^{fin} \cup \underline{\text{true}}] \\
&\sim\!\sim R^{fin*}
\end{aligned}
$$

(since $X^{fin} \in Auxv$, by the correspondence $\alpha Y[R;Y \cup \underline{\text{true}}] \sim\!\sim R^{*}$ for any R)

$$
\begin{aligned}
R^{\dagger inf} &\sim\!\sim \mu X[R;X \cup \underline{\text{true}}]^{inf} \sim\!\sim \mu X^{inf}[(R;X \cup \underline{\text{true}})^{inf}[R^{\dagger fin}/X^{fin}]] \\
&\sim\!\sim \mu X^{inf}[((R;X)^{inf} \cup \underline{\text{true}}^{inf})[R^{\dagger fin}/X^{fin}]] \\
&\sim\!\sim \mu X^{inf}[(R^{inf} \cup R^{fin};X^{inf})[\ldots]] \sim\!\sim (X^{fin} \text{ not in } (\ldots)) \\
&\sim\!\sim \mu X^{inf}[R^{inf} \cup R^{fin};X^{inf}] \\
&\sim\!\sim R^{fin*}\,R^{inf} \cup R^{fin\omega}
\end{aligned}
$$

(since $X^{inf} \in \mathit{Stmv}$, we can apply the correspondence $\mu X[R_1;X \cup R_2] \sim\!\sim R_1^*;R_2 \cup R_1^\omega$, for any R_1, R_2)

The remainder of this section is devoted to the proof that definition 4.2 is indeed the right one. We shall show that, for each *closed* S, $A(S^{fin}) = M(S)^{fin}$, $T(S^{inf}) = M(S)^{inf}$. (For S' not closed, the claim has to be somewhat refined, as will become clear from the subsequent discussion.) We first need the following simple property of fixed points:

LEMMA 4.3. *Let* $f \in [C \to C]$, $g \in [C \to_s C']$, $h \in C' \to_m C'$. *Assume that, for all* x,

(4.2) $\quad g(f(x)) = h(g(x))$.

Then μh *exists, and*

(4.3) $\quad g(\mu f) = \mu h$.

PROOF. Putting $x = \mu f$ in (4.2) we obtain $g(\mu f) = h(g(\mu f))$. Thus, $g(\mu f)$ is a fixed point of h. We shall show that it is, in fact, the *least* fixed point of h. Let x_0 be *any* fixed point of h. We shall show that $g(\mu f) \sqsubseteq x_0$. We use that $\mu f = \bigsqcup_i f^i(\perp)$. By continuity of g it is sufficient to prove $(*): g(f^i(\perp)) \sqsubseteq x_0$, for all i. The case i = 0 follows from strictness of g. Now assume $(*)$, to show $g(f^{i+1}(\perp)) \sqsubseteq x_0$. By (4.2), $g(f^{i+1}(\perp)) = g(f(f^i(\perp)) = h(g(f^i(\perp)) \sqsubseteq$ (by monotonicity of h and $(*)$) $h(x_0) = x_0$. $\square$

Remark. A similar result is used in [1]. The lemma is a slight extension of exercise 5-3 of [4], in that h is assumed monotonic rather than continuous.

Below, we shall need a simple generalization of lemma 4.3 to the case of systems of mappings g_1, g_2, h_1, h_2:

COROLLARY 4.4. *Let* $f \in [C \to C]$, $g_i \in [C \to_s C_i]$, $i = 1,2$, $h_i \in C_1 \to_m (C_2 \to_m C_i)$, $i = 1,2$. *Then from*

$$g_1(f(x)) = h_1(g_1(x))(g_2(x))$$
$$g_2(f(x)) = h_2(g_1(x))(g_2(x))$$

it follows that

$$g_1(\mu f) = \mu[\lambda y.h_1(y)(g_2(\mu f))]$$
$$g_2(\mu f) = \mu[\lambda z.h_2(g_1(\mu f))(z)].$$

PROOF. Easy extension of the proof of lemma 4.3. □

The property of least fixed points as stated in lemma 4.3 is at the heart of a number of results concerning recursive procedures. More specifically, it can be used to justify a variety of syntactic transformations (such as *fin* and *inf* studied here) by connecting them to one or more semantic transformations such as the mappings g, g_1, g_2 encountered above. The general pattern of this connection is the following: Let $Synt_1$, $Synt_2$ be two syntactic classes with typical elements D,..., F,..., respectively. Each of them has certain constructs we leave unspecified, furthermore classes of variables Var_1, Var_2, with typical elements x,..., and y,..., respectively, and μ-forming operators μx[...] and μy[...]. Thus, we assume a syntax

$$D ::= \ldots | x | \mu x[D]$$
$$F ::= \ldots | y | \mu y[F].$$

We also assume that substitutions D[D'/x], F[F'/y] are defined in the usual manner. Next we assume that the elements of $Synt_1$, $Synt_2$ obtain meanings through valuations $\mathcal{D}, \mathcal{F}$- with respect to the usual environment E; its precise definition as E_D or E_F is left to the reader - yielding results in cpo's $(\xi\in)\ K_D$, $(\eta\in) K_F$, respectively. More specifically let

$$\mathcal{D}\colon Synt_1 \to (E\to K_D)$$
$$\mathcal{F}\colon Synt_2 \to (E\to K_F)$$

be defined for variables and μ-terms in the usual way:

$$\mathcal{D}(x)(\varepsilon) = \varepsilon(x),\ \mathcal{F}(y)(\varepsilon) = \varepsilon(y),\ \text{and}$$

$$(4.4)\qquad \begin{aligned} \mathcal{D}(\mu x[D])(\varepsilon) &= \mu[\lambda\xi.\mathcal{D}(D)(\varepsilon\{\xi/x\})] \\ \mathcal{F}(\mu y[F])(\varepsilon) &= \mu[\lambda\eta.\mathcal{F}(F)(\varepsilon\{\eta/y\})]. \end{aligned}$$

(In (4.4), we take least fixed points with respect to the ordering in $[K_D \to K_D]$, $[K_F \to K_F]$ respectively.) Furthermore, we require that $\mathcal{D}$, $\mathcal{F}$ satisfy the conditions

$$(4.5)\qquad \begin{aligned} \mathcal{D}(D[D'/x])(\varepsilon) &= \mathcal{D}(D)(\varepsilon\{\mathcal{D}(D')(\varepsilon)/x\}) \\ \mathcal{F}(F[F'/y])(\varepsilon) &= \mathcal{F}(F)(\varepsilon\{\mathcal{F}(F')(\varepsilon)/y\}) \end{aligned}$$

The reader should observe that all we do here is to give a somewhat abstract version of the properties of *Stat, Auxs*, with valuations *M*, *A*.

Now let "~" be a (syntactic) mapping: $Synt_1 \to Synt_2$. Usually, it is reasonably easy in a specific instance of a transformation "~" to establish how it should be defined for the non-recursive case, and one would expect the "~" definition for μ-constructs to be the more difficult part. However, it was a pleasant surprise for us to discover that, on the contrary, once one has found the appropriate definition for the non-recursive case, it is possible - under the quite general assumptions mentioned above - to provide a standard treatment of the case of a μ-term.

Let us assume that "~" satisfies the general property that, for each $x \in Var_1$, $\tilde{x}$ is an element of Var_2, and that, moreover, "~" is an injection. We also require for each D that $\tilde{D}$ contains no free variables other than those induced by "~" from the free variables of D. Let us furthermore postulate that "~" is defined for a μ-term by

$$(4.6)\qquad \mu x[D]^{\sim} \equiv \mu\tilde{x}[\tilde{D}]$$

We shall show that (4.6) is satisfactory in the following sense: Often, we want to justify the definition of "~" by showing that it induces a certain semantic property, say κ, which can be seen as mapping between the semantic domains, i.e., we take $\kappa\colon K_D \to K_F$. (In the example of *fin*, the semantic counterpart is the mapping $fin\colon \phi \mapsto \phi^{fin} = \lambda\sigma.\phi(\sigma)^{fin}$.) We then wish to establish commutativity of the diagram

$$\begin{tikzcd} \mathit{Synt}_1 \arrow[r, "\sim"] \arrow[d, "\mathcal{D}"'] & \mathit{Synt}_2 \arrow[d, "\mathcal{F}"] \\ K_D \arrow[r, "\kappa"'] & K_F \end{tikzcd}$$

The commutativity requirement for *variables* specializes to $(*)$: $\kappa(\varepsilon(x)) = \varepsilon(\tilde{x})$ (since $\mathcal{D}(x)(\varepsilon) = \varepsilon(x)$, $\mathcal{F}(y)(\varepsilon) = \varepsilon(y)$). In case ε satisfies $(*)$ for all x, we call ε *consistent*.

In order to analyze the relationship between "$\sim$" and "κ", is particular for μ-terms, we introduce two operators Φ_D, Ψ_D in the following way: Let, for $D \in \mathit{Synt}_1$, $\mathit{var}(D) = \{x_1,\ldots,x_n\}$ be the set of *free* variables of D, and let $\{\tilde{x}_1,\ldots,\tilde{x}_n\}$ be the free variables of $\tilde{D}$. Let $\vec{x}$ abbreviate $x_1,\ldots,x_n$ (in some arbitrary, but fixed order), and let $\vec{\xi} = \xi_1,\ldots,\xi_n$, $\vec{\eta} = \eta_1,\ldots,\eta_n$. We now define $\Phi_D: K_D^n \to K_D$, $\Psi_D: K_F^n \to K_F$ by

$$\Phi_D = \lambda\vec{\xi}.\mathcal{D}(D)(\varepsilon\{\xi_i/x_i\}_i)$$

$$\Psi_D = \lambda\vec{\eta}.\mathcal{F}(\tilde{D})(\varepsilon\{\eta_i/\tilde{x}_i\}_i)$$

and we investigate whether the relationship

$$(4.7) \qquad \kappa(\Phi_D(\vec{\xi})) = \Psi_D(\kappa(\vec{\xi}))$$

holds for all $\vec{\xi}$. Indeed for consistent ε, taking $\xi_i = \varepsilon(x_i)$, $\eta_i = \kappa(\varepsilon(x_i))$, $i = 1,\ldots,n$, and using that $\kappa(\varepsilon(x_i)) = \varepsilon(\tilde{x}_i)$, $i = 1,\ldots,n$, $\varepsilon\{\varepsilon(x_i)/x_i\}_i = \varepsilon\{\varepsilon(\tilde{x}_i)/\tilde{x}_i\}_i = \varepsilon$, we see that (4.7) is equivalent with

$$(4.8) \qquad \kappa(\mathcal{D}(D))(\varepsilon) = \mathcal{F}(\tilde{D})(\varepsilon),$$

which is the same as the commutativity of the diagram above. For example, for "$\sim$" and "κ" instantiated to the syntactic and semantic *fin*, and with the natural correspondence between $Synt_1$ and $Stat$, etc., (4.7) reduces to the claim $M(S)(\varepsilon)^{fin} = A(S^{fin})(\varepsilon)$ - where consistency now means that $\varepsilon(X^{fin}) = \varepsilon(X)^{fin}$.

In order to prove (4.7) in the general case, one proceeds by induction on the complexity of D. One would expect the non-recursive cases of such an induction to be reasonably easy, whereas the difficult case would be that of recursion. However, we claim that - provided that the various properties of $\sim$, κ, $\mathcal{D}$ and $\mathcal{F}$ listed above are satisfied - the μ-case of the induction is automatically obtained. In fact the following theorem holds.

THEOREM 4.5. *Assume that* $\sim$, κ, $\mathcal{D}$, $\mathcal{F}$ *satisfy the properties mentioned above. (In particular,* (4.4) *to* (4.6) *hold.) Assume, moreover, that* $\kappa \in [K_D \to_s K_F]$, $\Phi_D \in [K_D^n \to K_D]$, $\Psi_D \in K_F^n \to_m K_F$. *Then, if* (4.7) *holds for* $D \equiv D_0$ *(and* $n=k+1$*), then it holds for* $D \equiv \mu x[D_0]$ *(and* $n=k$*)*.

PROOF. By an easy extension of lemma 4.3 we obtain that if, for all $\xi_1,\ldots,\xi_n$,

$$\kappa(\Phi_D(\xi_1)\ldots(\xi_n)) = \Psi_D(\kappa(\xi_1))\ldots(\kappa(\xi_n))$$

then, for all $\xi_1,\ldots,\xi_{n-1}$,

$$\kappa(\mu[\lambda\xi.\Phi_D(\xi_1)\ldots(\xi_{n-1})(\xi)]) = \mu[\lambda\eta.\Psi_D(\kappa(\xi_1))\ldots(\kappa(\xi_{n-1}))(\eta)]. \tag{4.9}$$

We now show that if (4.7) holds for $D \equiv D_0$ (and $n=k+1$) then it holds for $D \equiv \mu x[D_0]$ (and $n=k$). Let $D \equiv \mu x[D_0]$. By the definition of Φ_D and Ψ_D we have to show that, for all $\xi_1,\ldots,\xi_n$,

$$\kappa(\mathcal{D}(\mu x[D_0])(\varepsilon\{^{\xi_i}/x_i\}_i) = \mathcal{F}(\mu x[D_0]^{\sim})(\varepsilon\{^{\kappa(\xi_i)}/\tilde{x}_i\}_i).$$

We only consider the subcase that $x \not\equiv x_1,\ldots,x_n$, andleave the other subcase to the reader. By applying (4.4) and (4.6) to the left-hand side (lhs) and right-hand side (rhs), respectively, what we have to prove reduces to

$$\kappa(\mu[\lambda\xi.\mathcal{D}(D_0)(\varepsilon\{^{\xi_i}/x_i\}\{^{\xi}/x\})]) = \mathcal{F}(\mu\tilde{x}[\tilde{D}_0])(\varepsilon\{^{\kappa(\xi_i)}/\tilde{x}_i\}_i).$$

By the assumption we know that (4.7) holds for Φ_{D_0}, Ψ_{D_0}, and we can rewrite the lhs using (4.9). Also applying the definition of $\mathcal{F}$ to the rhs, we obtain

$$\text{lhs} = \mu[\lambda\eta.\mathcal{F}(\tilde{D}_0)(\varepsilon\{^{\kappa(\xi_i)}/\tilde{x}_i\}_i\{^{\eta}/\tilde{x}\})],$$

$$\text{rhs} = \mu[\lambda\eta.\mathcal{F}(\tilde{D}_0)(\varepsilon\{^{\kappa(\xi_i)}/\tilde{x}_i\}_i\{^{\eta}/\tilde{x}\})].$$

We see that lhs and rhs are identical, thus completing the proof. □

Based on corollary 4.4, we can formulate a direct generalization of this theorem in

COROLLARY 4.6. *Assume the following framework*:
$(D\in)\mathit{Synt}, (F\in)\mathit{Synt}_1, (G\in)\mathit{Synt}_2, (x\in)\mathit{Var}, (y\in)\mathit{Var}_1, (z\in)\mathit{Var}_2$,
$D ::= \ldots|x|\mu x[D]$, $F ::= \ldots|y|\mu y[F]$, $G ::= \ldots|z|\mu z[G]$, $\mathcal{D}(x)(\varepsilon) = \varepsilon(x)$,
$\mathcal{D}(\mu x[D]) = \mu[\lambda\xi.\mathcal{D}(D)(\varepsilon\{^{\xi}/x\})]$ *and similarly for* $\mathcal{F}$, $\mathcal{G}$, $\mathcal{D}(D[^{D'}/x])(\varepsilon) =$
$= \mathcal{D}(D)(\varepsilon\{^{\mathcal{D}(D')(\varepsilon)}/x\})$, *and similarly for* $\mathcal{F}$, $\mathcal{G}$, $\tilde{}^{\,i}: \mathit{Synt} \to \mathit{Synt}_i$, $i = 1,2$,
$\tilde{x}^i \in \mathit{Var}_i$, $\sim i$ *are injections*, $i = 1,2$,

$$(4.10)\qquad \begin{aligned} \mu x[D]^{\sim 1} &\equiv \mu\tilde{x}^1[\tilde{D}^1[\mu x[D]^{\sim 2}/\tilde{x}^2]] \\ \mu x[D]^{\sim 2} &\equiv \mu\tilde{x}^2[\tilde{D}^2[\mu x[D]^{\sim 1}/\tilde{x}^1]] \end{aligned}$$

$\Phi_D = \lambda\vec{\xi}.\mathcal{D}(D)(\varepsilon\{^{\xi_i}/x_i\}_i)$, $\Psi_D^1 = \lambda\vec{\eta}.\lambda\vec{\zeta}.\mathcal{F}(\tilde{D}^1)(\varepsilon\{^{\eta_i}/\tilde{x}_i^1\}\{^{\zeta_i}/\tilde{x}_i^2\}_i)$, *and similarly for* Ψ_D^2, $\kappa_1 \in [K_D \to_s K_F]$, $\kappa_2 \in [K_D \to_s K_G]$, $\Phi_D \in [K_D^n \to K_D]$, $\Psi_D^1 \in K_F^n \to_m (K_G^n \to_m K_F)$, *and similarly for* Ψ_D^2. *Then, if, for all* ξ,

$$\kappa_1(\Phi_D(\vec{\xi})) = \Psi_D^1(\kappa_1(\vec{\xi}))(\kappa_2(\vec{\xi}))$$

$$\kappa_2(\Phi_D(\vec{\xi})) = \Psi_D^2(\kappa_1(\vec{\xi}))(\kappa_2(\vec{\xi}))$$

holds for $D \equiv D_0$ (*and* $n=k+1$), *then it holds for* $D \equiv \mu x[D_0]$ (*and* $n=k$).

PROOF. Follows the same lines as the proof of theorem 4.5, now based on the semantic property of corollary 4.4. □

We are finally ready for the proof of the main result of the paper.

Analogous to the above definitions, we call ε consistent if, for all X, $\varepsilon(X)^{fin} = \varepsilon(X^{fin})$, and $\varepsilon(X)^{inf} = \varepsilon(X^{inf})$.

THEOREM 4.7. *For all consistent* ε,

$$M(S)(\varepsilon)^{fin} = A(S^{fin})(\varepsilon)$$
$$M(S)(\varepsilon)^{inf} = T(S^{inf})(\varepsilon).$$

PROOF. Induction on the complexity of S. First we consider the case that S is not a μ-term.

a. $S \equiv x:=s,\ b,\ \Delta$. Trivial.

b. $S \equiv S_1;S_2$. This case follows since, for all τ
 (i) $\hat{\phi}(\tau)\backslash\{\bot\} = \hat{\phi}(\tau\backslash\{\bot\})\backslash\{\bot\}$, hence $\lambda\sigma.\hat{\phi}_2(\sigma))^{fin} = \lambda\sigma.\hat{\phi}_2^{fin}(\phi_1^{fin}(\sigma))$
 (ii) $\hat{\phi}(\tau)^{inf} = \tau^{inf} \cup \hat{\phi}^{inf}(\tau^{fin})$, hence
 $\lambda\sigma.\hat{\phi}_2(\phi_1(\sigma))^{inf} = \lambda\sigma.[\phi_1^{inf}(\sigma) \cup \hat{\phi}_2^{inf}(\phi_1^{fin}(\sigma))]$

c. $S \equiv S_1 \cup S_2$. Obvious

d. $S \equiv X$. Follows from the consistency requirement.

e. $S \equiv \mu X[S_0]$. Follows from corollary 4.6. We take $\sim 1 = fin$, $\sim 2 = inf$, $\kappa_1 = fin$, $\kappa_2 = inf$ (syntactic and semantic *fin* and *inf*, respectively), $Synt = Stat$, $Synt_1 = Auxs$, $Synt_2 = Exts$, $Var = Stmv$, $Var_1 = Auxv$, $Var_2 = Stmv$, $D = M$, $F = A$, $G = T$, $K_D = (M,\sqsubseteq)$, $K_F = (M,\subseteq)$, $K_G = (M,\sqsubseteq)$. Strictness of κ_1 follows from $\{\bot\}\backslash\{\bot\} = \emptyset$ (the least element of $(T,\subseteq)$), and continuity from $(\bigsqcup_i \tau_i)\backslash\{\bot\} = \bigcup_i(\tau_i\backslash\{\bot\})$. Strictness of κ_2 follows from $\{\bot\}\backslash\{\bot\}^{fin} = \{\bot\}$, and continuity from $(\bigsqcup_i \tau_i)\backslash(\bigsqcup_i \tau_i\backslash\{\bot\}) = \bigsqcup_i (\tau_i\backslash(\tau_i\backslash\{\bot\}))$, i.e., from $\bot \in \bigsqcup_i \tau_i$ iff $\bot \in \tau_i$ for all i. Finally, we verify whether (4.10) is satisfied, i.e., whether

$$\mu X[S]^{fin} \equiv \alpha X^{fin}[S^{fin}[\mu X[S]^{inf}/X^{inf}]]$$
$$\mu X[S]^{inf} \equiv \mu X^{inf}[S^{inf}[\mu X[S]^{fin}/X^{fin}]]. \qquad (4.11)$$

Observing that X^{inf} does not occur free in S^{fin}, we see that (4.11) reduces to

$$\mu X[S]^{fin} \equiv \alpha X^{fin}[S^{fin}]$$
$$\mu X[S]^{inf} \equiv \mu X^{inf}[S^{inf}[\mu X[S]^{fin}/X^{fin}]]$$

which is indeed the form of definition 4.2. □

We have thus completed the justification of definition 4.2 on the basis of a general argument concerning properties of recursive procedures.

5. SYSTEMS OF RECURSIVE PROCEDURES AND NIVAT's THEOREM

We discuss the relationship between the results of the previous section and a theorem of Nivat on infinite words generated by a context free grammar. We begin with a reformulation of our theorem for a language which has systems of (simultaneously declared) recursive procedures rather than the μ-terms of the preceding sections. Since the structure of a system of recursive procedures closely resembles that of a context free grammar, we thus obtain a framework facilitating the comparison with Nivat's result. We redefine syntax and semantics of our language *Stat* as follows:

DEFINITION 5.1 (syntax and semantics of a language with systems of recursive procedures, *fin* and *inf*).

a. Let $(P\in)$ *Pvar* be the set of *procedure variables*. Let $(S\in)$ *Stat* be redefined by

$$S ::= x:=s \mid b \mid \Delta \mid S_1;S_2 \mid S_1 \cup S_2 \mid P$$

and let $(R\in)$ *Prog* be the class of *programs* of the form $\langle\langle P_i \Leftarrow S_i\rangle_i \mid S\rangle$: A program R is a pair consisting of a set of declarations $P_i \Leftarrow S_i$, $i = 1,\ldots,n$, and a (main) statement S.

b. Let E: *Pvar* $\rightarrow$ M be as usual, and let N: *Prog* $\rightarrow$ (E$\rightarrow$M) be defined by

$$N(\langle\langle P_i \Leftarrow S_i\rangle_i \mid S\rangle)(\varepsilon) = M(S)(\varepsilon\{\phi_i/P_i\}_i)$$

where M is as before for S not a procedure variable, $M(P)(\varepsilon) = \varepsilon(P)$ and $\phi_i = \mu_i[\Phi_1,\ldots,\Phi_n]$, with $\mu_i[\ldots]$ denoting the i-th component of the simultaneous least fixed point of the n-tuple of continuous functions $\Phi_1,\ldots,\Phi_n$, and $\Phi_j = \lambda\phi_1'.\ldots.\lambda\phi_n'.M(S_j)(\varepsilon\{\phi_i'/P_i\}_i)$.

c. Let $(A\in)$ *Auxs* and $(T\in)$ *Exts* be defined as before for the non-procedure cases and let $(Q\in)$ *Auxv* be the set of auxiliary procedure variables.

Programs $<<Q_j \Leftarrow A_j>_j \mid A>$ and $<<P_i \Leftarrow T_i>_i,<Q_j \Leftarrow A_j>_j \mid T>$ obtain meaning with valuations $\mathcal{B}$ and $\mathcal{U}$ defined by

$$\mathcal{B}(<<Q_j \Leftarrow A_j>_j|A>)(\varepsilon) = \mathcal{A}(A)(\varepsilon\{\psi_j/Q_j\}_j)$$
$$\mathcal{U}(<<P_i \Leftarrow T_i>_i,<Q_j \Leftarrow A_j>_j|T>)(\varepsilon) = \mathcal{T}(T)(\varepsilon\{\phi_i/P_i\}_i\{\psi_j/Q_j\}_j)$$

where $\mathcal{A}(A)(\varepsilon)$ and $\mathcal{T}(T)(\varepsilon)$ are defined in the natural way for A,T not a procedure variable, and, moreover, $\mathcal{A}(Q)(\varepsilon) = \varepsilon(Q), \mathcal{T}(P)(\varepsilon) = \varepsilon(P)$, and

$$\psi_j = \mu_{\subseteq,j}[\Psi_1,\ldots,\Psi_n],\ \Psi_j = \lambda\psi_1' \ldots \lambda\psi_n'.\mathcal{A}(A_j)(\varepsilon\{\psi_i'/Q_i\}_i)$$

$$\phi_i = \mu_{\sqsubseteq,i}[\Phi_1,\ldots,\Phi_n],\Phi_i = \lambda\phi_1' \ldots \lambda\phi_n'.\mathcal{T}(T_i)(\varepsilon\{\phi_i'/P_i\}_i\{\psi_j/Q_j\}_j)$$

d. We define *fin* and *inf* by

$$<<P_i \Leftarrow S_i>_i \mid S>^{fin} \equiv <<P_i^{fin} \Leftarrow S_i^{fin}>_i \mid S^{fin}>$$
$$<<P_i \Leftarrow S_i>_i \mid S>^{inf} \equiv <<P_i^{inf} \Leftarrow S_i^{inf}>_i,<P_i^{fin} \Leftarrow S_i^{fin}>_i \mid S^{inf}>$$

where S^{fin}, S^{inf} are defined as usual for S not a procedure variable, $P^{fin} \in Auxv$, and $P^{inf} \in Pvar$.

Remarks

1. In this section, R ranges over *Prog* rather than over *Regs*.
2. Note that, by the definitions of *fin* and *inf*, P^{inf} does not occur in S^{fin}; hence, again (as with definition 4.2) $S^{fin} \in Auxs$.
3. Note that in the definition of $\mathcal{N}(<<P_i \Leftarrow S_i>_i|S>)$, least fixed points are taken with respect to "$\sqsubseteq$", and in that of $\mathcal{B}(<<Q_j \Leftarrow A_j>_j|A>)$, least fixed points are taken with respect to "$\subseteq$". The former least fixed points are least upper bounds of chains $S^{(k)}$ defined inductively by $S^{(0)} \equiv \Omega$, $S^{(k+1)} \equiv S[S_i^{(k)}/P_i]_i$, whereas the latter are least upper bounds of chains $A^{(k)}$ defined by $A^{(0)} \equiv \underline{false}$, $A^{(k+1)} \equiv A[A_j^{(k)}/Q_j]_j$. Finally, in the definition of $\mathcal{U}(<<P_i \Leftarrow T_i>_i,<Q_j \Leftarrow A_j>_j|T>)$ a mixture of the two orderings is used. Since the P_i do not occur in the A_j, the definition does not have to be fully simultaneous in the P_i,Q_j together: in the definition of the ϕ_i, we may assume the ψ_j to be already determined.

Example. Let C_i, $i = 1,\ldots,$ stand for arbitrary statements without occurrences of procedure variables, and let R be defined by

$$P_1 \Leftarrow C_1;P_1;P_2 \cup C_2;P_2;P_1;C_3 \cup C_4,$$
$$P_2 \Leftarrow C_5;P_2;C_6 \cup C_7;P_2;P_2 \mid P_1>.$$

Then, using that $C_i^{fin} \equiv C_i$, $C_i^{inf} \equiv \underline{\text{false}}$, we obtain

$$R^{fin} \equiv <P_1^{fin} \Leftarrow C_1;P_1^{fin};P_2^{fin} \cup C_2;P_2^{fin};P_1^{fin};C_3 \cup C_4,$$
$$P_2^{fin} \Leftarrow C_5;P_2^{fin};C_6 \cup C_7;P_2^{fin};P_2^{fin} \mid P_1^{fin}>$$

and

$$R^{inf} \equiv <P_1^{fin} \Leftarrow \ldots,\ P_2^{fin} \Leftarrow \ldots \text{ (as above)},$$
$$P_1^{inf} \Leftarrow C_1;P_1^{inf} \cup C_1;P_1^{fin};P_2^{inf} \cup C_2;P_2^{inf} \cup C_2;P_2^{fin};P_1^{inf},$$
$$P_2^{inf} \Leftarrow C_5;P_2^{inf} \cup C_7;P_2^{inf} \cup C_7;P_2^{fin};P_2^{inf} \mid P_1^{inf}>.$$

Observe that programs R and R^{fin} are syntactically isomorphic (just as S and S^{fin} in section 4). The difference between them lies only in the way their meaning is defined.

We now state Nivat's theorem. Consider a context free grammar $G = (V_N,V_T,P)$, where $V_N = \{X_1,\ldots\}$, $V_T = \{a,\ldots\}$ are the alphabets of nonterminal and terminal symbols, and P is the set of production rules $X_i \rightarrow M_i$, M_i a finite set of words $\alpha \in (V_N \cup V_T)^*$. (We have no reason here to single out a start symbol.) Let, for finite or infinite terminal words x', x", $x' < x''$ denote that x' is a prefix of x". Let a finite derivation $\alpha \overset{*}{\Rightarrow}_i \alpha'$ be defined in the usual way. Moreover, we say that, for infinite x, $X \overset{\omega}{\Rightarrow} x$ (the nonterminal X derives the infinite word $x \in V_T^{\omega}$ in an infinite number of steps) whenever there exist finite prefixes x_i, $i = 1,2,\ldots,$ of the infinite word x such that, for all i, $X \overset{i}{\Rightarrow} x_i\alpha_i$ for some α_i, and $x_1 < x_2 <\ldots< x_i <\ldots< x = \bigvee_i x_i$, i.e., x is the least upper bound of the <-chain $<x_i>_i$ with respect to the prefix ordering. Let $L(G,X_i)^f$ stand for the set of finite words generated by X_i, and $L(G,X_i)^{\omega}$ for the set of infinite words generated by X_i. We then have

<u>THEOREM 5.2</u> ([13]). *Let* G, *with production rules* $P = \langle X_i \to M_i\rangle_i$ *be a context free grammar as described above, and let* G^ω *be the context free grammar* $(\bar{V}_N, V_N \cup V_T, \bar{P})$, *where* $\bar{V}_N = \{\bar{X}_1,\ldots\}$, *and* $\bar{P}$ *is the set of production rules* $\langle \bar{X}_i \to \bar{M}_i\rangle_i$, *where the* $\bar{M}_i$ *are finite subsets of* $(\bar{V}_N \cup V_N \cup V_T)^*$ *defined by*

$$\bar{M}_i = \{\alpha\bar{X} \mid \alpha X_j \beta \in M_i \text{ for some } \alpha,\beta \in (V_N \cup V_T)^*, \quad X_j \in V_N\}.$$

Let L_j^f *abbreviate* $L(G,X_j)^f$, *and let, for any language* L *over the terminals* $V_N \cup V_T$, $L[L_j^f/X_j]$ *denote the result of substituting the languages* L_j^f *for the (terminal!)* X_j *in the words of* L *(with the precaution that substitution in an infinite word yields an infinite word; this is made precise* [13]). *We then have, for* $i = 1,\ldots,n$,

$$L(G,X_i)^\omega = L(G^\omega,\bar{X}_i)[L_j^f/X_j]_j.$$

PROOF. See [13]. □

Example. Consider the context free grammar with productions P:

$$X_1 \to a_1X_1X_2 \mid a_2X_2X_1a_3 \mid a_4$$
$$X_2 \to a_5X_2a_6 \mid a_7X_2X_2.$$

For the set of productions $\bar{P}$ we obtain by the construction of the theorem

$$\bar{X}_1 \to a_1\bar{X}_1 \mid a_1X_1\bar{X}_2 \mid a_2\bar{X}_2 \mid a_2X_2\bar{X}_1$$
$$\bar{X}_2 \to a_5\bar{X}_2 \mid a_7\bar{X}_2 \mid a_7X_2\bar{X}_2.$$

We observe a remarkable similarity between the system $\bar{P}$ and the definition of R^{inf} in the example following definition 5.1. In fact, we shall formulate a commutativity result which makes this observation precise. First we need a number of preparations. Each program R of the form $\langle\langle P_i \Leftarrow S_i\rangle_i \mid P_k\rangle$ can be viewed as a grammar - with start symbol P_k - generating (finite or infinite) words over the alphabet of "terminals" $x:=s, b, \Delta$ in a natural way. E.g., the program $R_0 \equiv \langle P \Leftarrow C_i;P;C_2 \cup C_3 \mid P\rangle$ determines the language

$C_1^{\omega} \cup \{C_1^n;C_3;C_2^n \mid n \geq 1\}$ (with C_i, as before, statements without free occurrences of P). Let us, till the end of this section and essentially without lack of generality, restrict attention to programs of the form $R \equiv \langle\langle P_i \Leftarrow S_i\rangle_i | S\rangle$, where $S \equiv P_k$ for some k, $1 \leq k \leq n$, and each of the S_i is of the form $S_i \equiv S_{i1} \cup \ldots \cup S_{in_i}$, with each S_{ij} of the form $C_1;P_{k_1};C_2;P_{k_2};\ldots;P_{k_r};C_{r+1}$, and, conventionally, with true taking the role of the empty word. It should be clear how such an R can be seen as a grammar generating (finite or infinite) sequences of (elementary) statements as indicated by the above example of program R_0. For each such R, its associated language is denoted by $L(R)$. Furthermore, for a program $\bar{R}$ of the form $\langle\langle P_i \Leftarrow T_i\rangle_i, \langle Q_j \Leftarrow A_j\rangle \mid P_k\rangle$ - with analogous restrictions on the form of the T_i, A_j - we have as associated language $L(\bar{R})$ all words which can be derived starting from P_k, where for the nonterminals P_i finite or infinite derivations are used, and for the Q_j only finite derivations. The next step consists in the observation that the mapping $M \mapsto \bar{M}$ (as described in the statement of the theorem) is isomorphic to the mapping $S \mapsto S^{inf}$, where occurrences $X_i, \bar{X}_i$ in $\bar{M}$ correspond to occurrences of P_i^{fin}, P_i^{inf} in S^{inf}. For example, $aX_1X_2 \mapsto a\bar{X}_1 \cup aX_1\bar{X}_2$, whereas $C;P_1;P_2 \overset{inf}{\mapsto} C;P_1^{inf} \cup C;P_1^{fin};P_2^{inf}$. Finally, we observe the following: The expression $L(G^{\infty},\bar{X}_i)[{}^{Lf}_j/X_j]_j$ occurring in the statement of theorem 5.2 can be rewritten as $L(G^{\omega} \cup G^{f},\bar{X}_i)$, where G^f indicates that for the nonterminals X_j from G_j only finite derivations are allowed. Putting all these observation together, we obtain the following theorem:

<u>THEOREM 5.3</u>. *Let* $R \equiv \langle\langle P_i \Leftarrow S_i\rangle_i \mid P_k\rangle$ *be a program satisfying the above constraints. The diagram*

$$\begin{tikzcd}
R \arrow[r, "inf"] \arrow[d, "L"'] & R^{inf} \arrow[d, "L"] \\
L(R) \arrow[r, "\omega"'] & L(R)^{\omega} = L(R^{inf})
\end{tikzcd}$$

commutes.

PROOF. Let $D \equiv \langle P_i \Leftarrow S_i \rangle_i$, $D^{fin} \equiv \langle P_i^{fin} \Leftarrow S_i^{fin} \rangle_i$, $D^{inf} \equiv \langle P_i^{inf} \Leftarrow S_i^{inf} \rangle$. Then

$$L(\langle D|P_k \rangle)^{\omega} = \text{(by Nivat's theorem and the isomorphism mentioned above)}$$

$$L(\langle D^{inf}|P_k^{inf}\rangle)[L(\langle D^{fin}|P_j^{fin}\rangle)/P_j^{fin}]_j = L(\langle D^{inf} \cup D^{fin}|P_k^{inf}\rangle = L(R^{inf}).$$

□

This concludes our discussion of the relationship between infinite computations and languages with infinite words.

6. APPLICATIONS TO WEAKEST PRECONDITIONS

In this section, we discuss a number of applications of the proof techniques presented in section 4. In particular, we obtain a variety of results concerning weakest preconditions - mostly for regular statements - including many of those described in Chapter 8 of [4].

We first state an auxiliary result which is a variation on Lemma 4.3:

LEMMA 6.1. *Let* C *be a* cpo, C' *a complete lattice,* $f \in [C \to C]$ *and* $g: C \to C'$ *an antistrict and anticontinuous function. I.e., for* $\top$ *the greatest element of* C', $g(\bot) = \top$ *and, for each ascending chain* $\langle x_i \rangle_i$ *in* C', $g(\bigsqcup_i x_i) = \bigsqcap_i g(x_i)$.) *Let* $h: C' \to_m C'$. *Then from* $g \circ f = h \circ g$ *it follows that* νh *exists and that* $\nu h = g(\mu f)$ *holds.*

PROOF. Similar to the proof of Lemma 4.3. □

We leave to the reader statement and proof of a theorem which expresses the corresponding variation for theorem 4.5. The main changes are that the semantic mapping κ is now required to be antistrict and anticontinuous, and that definition (4.6)is replaced by

$$(6.1) \qquad \mu x[D]^{\sim} \equiv \nu\tilde{x}[\tilde{D}]$$

where the prefix $\nu\tilde{x}[\ldots]$ denotes the *greatest* fixed point operator.

We now introduce four notions of weakest precondition. They are presented through a variety of semantic composition formulae; later a syntactic notation corresponding to the four semantic notions is proposed. Let $(\pi\in)\Pi$ be the set of predicates, defined as $\Pi = \Sigma \rightarrow_{ss} \{tt,ff\}$, where $\rightarrow_{ss}$ denotes functions π such that $\pi(\delta) = \pi(\bot) = ff$. Let $\{tt,ff\}$ be ordered by $ff \sqsubseteq tt$, and let $\pi_1 \sqsubseteq \pi_2$ hold iff $\pi_1(\sigma) \sqsubseteq \pi_2(\sigma)$ for all $\sigma \in \Sigma$. Observe that it is immediate that Π is a complete lattice.

<u>DEFINITION 6.2</u>. For $\tau \in T$, $\pi \in \Pi$, $\phi \in M$ we put

a. $\pi[\tau] \iff \pi(\sigma)$ holds for all $\sigma \in \tau$

$\pi\{\tau\} \iff \pi(\sigma)$ holds for all $\sigma \in \tau\backslash\{\bot\}$

$\pi\langle\tau\rangle \iff \pi(\sigma)$ holds for all $\sigma \in \tau\backslash\{\delta\}$

$\pi(\tau) \iff \pi(\sigma)$ holds for all $\sigma \in \tau\backslash\{\bot,\delta\}$

b. $\phi[\pi] \iff \lambda\sigma.\pi[\phi(\sigma)]$

$\phi\{\pi\} \iff \lambda\sigma.\pi\{\phi(\sigma)\}$

$\phi\langle\pi\rangle \iff \lambda\sigma.\pi\langle\phi(\sigma)\rangle$

$\phi(\pi) \iff \lambda\sigma.\pi(\phi(\sigma))$.

<u>LEMMA 6.3</u>.

a. *The compositions* $\pi[\tau]$, $\pi\{\tau\}$, $\pi\langle\tau\rangle$, $\pi(\tau)$ *are all monotonic in* π

b. *The compositions* $\pi[\tau]$, $\pi\langle\tau\rangle$ *are strict and continuous in* τ

c. *The compositions* $\pi\{\tau\}$, $\pi(\tau)$ *are antistrict and anticontinuous in* τ.

<u>PROOF</u>. Direct from the definitions. □

Next, we introduce the syntactic class of *conditions* $(p\in)$ *Cond*, which extends the class of assertions (first order formulae) in two ways: Firstly, syntactic versions of the weakest precondition constructs as suggested by definition 6.2b are added, and secondly we introduce least fixed point and greatest fixed point forming operators. Let $(Z\in)$ *Cndv* be the class of condition variables. As in sections 2 to 4, R denotes a regular statement.

<u>DEFINITION 6.4</u> (syntax and semantics of conditions).

a. The class $(p\in)$ *Cond* is defined by

$$p ::= \underline{true} \mid \underline{false} \mid s_1=s_2 \mid \ldots \mid \neg p \mid p_1 \vee p_2 \mid \exists x[p] \mid R[p] \mid R\{p\} \mid R\langle p\rangle \mid R(p) \mid Z \mid \mu Z[p] \mid \nu Z[p]$$

where in the last two clauses p is required to be syntactically monotonic in Z (Z does not occur in the scope of an odd number of $\neg$-signs)

b. Let $E = Cndv \to \Pi$, and let R be as in section 2. The valuation C: $Cond \to (E \to \Pi)$ is defined by $C(p)(\varepsilon)(\delta) = C(p)(\varepsilon)(\perp) = ff$, and, for $\sigma \in \Sigma_0$,
$C(\underline{\text{true}})(\varepsilon) = \lambda\sigma.tt, \ldots, C(\exists x[p])(\varepsilon) = \lambda\sigma.\exists\alpha C(p)(\varepsilon)(\sigma\{^{\alpha}/x\})$,
$C(R[p])(\varepsilon) = R(R)[C(p)(\varepsilon)]$
$C(R\{p\})(\varepsilon) = R(R)\{C(p)(\varepsilon)\}$
$C(R\langle p\rangle)(\varepsilon) = R(R)\langle C(p)(\varepsilon)\rangle$
$C(R(p))(\varepsilon) = R(R)(C(p)(\varepsilon))$
$C(Z)(\varepsilon) = \varepsilon(Z)$
$C(\mu Z[p])(\varepsilon) = \mu[\lambda\pi.C(p)(\varepsilon\{^{\pi}/Z\})]$
$C(\nu Z[p])(\varepsilon) = \nu[\lambda\pi.C(p)(\varepsilon\{^{\pi}/Z\})]$

c. We put $\models p_1 = p_2$ whenever, for all $\varepsilon, \sigma, C(p_1)(\varepsilon)(\sigma) = C(p_2)(\varepsilon)(\sigma)$.

Remarks

1. A similar variety of weakest preconditions has been investigated in an operational setting by HAREL [8].
2. Clearly, we can now introduce four notions of correctness of a statement R (or in general, S) with respect to conditions p,q, viz. $[p]R[q]$ defined as $p \supset R[q], \ldots, (p)R(q)$ defined as $p \supset R(q)$.
3. A *fifth* weakest precondition could be based on the composition $\pi[\![\tau]\!] \iff \pi(\sigma)$ holds for some $\sigma \in \tau$. We shall not pursue this possibility here.

We are now sufficiently prepared for the main theorem of this section:

<u>THEOREM 6.5</u>. *For* Z *not free in* p,

a. $\models R^{\dagger}[p] = \mu Z[R[Z] \wedge p]$
b. $\models R^{\dagger}\{p\} = \nu Z[R\{Z\} \wedge p]$
c. $\models R^{\dagger}\langle p\rangle = \mu Z[R\langle Z\rangle \wedge p]$
d. $\models R^{\dagger}(p) = \nu Z[R(Z) \wedge p]$

<u>PROOF</u>. We only prove part b, the other cases being quite similar.
In order to be able to apply the theory of section 4, we slightly extend the class of regular statements as introduced before. Recall that a statement is called *closed* whenever it has no free occurrences of a statement variable.

We now put - for the duration of this proof only -

$$R::= x:=s\,|\,b\,|\,\Delta\,|\,R_1;R_2\,|\,R_1\cup R_2\,|\,R_1;X\cup R_2\,|\,\mu X[R_1;X\cup R_2]$$

where the R_1,R_2 on the right-hand side of the definition are required to be closed. (Thus, an extended regular statement has at most one free occurrence of (at most) one statement variable.) Let Regs stand for the class of extended regular statements. From section 2 it should be clear how to define $\mathcal{R}$: $\mathit{Regs} \to (E \to M)$; environments $\varepsilon(\in E)$ are now defined for both statement variables and condition variables. We next define a syntactic mapping: $\mathit{Regs} \to \mathit{Cond}$ - depending on a parameter p - which maps each statement R to a concertion written as {R} p (though similar to R{p}, it should for the moment be distinguished from it). For a statement variable X, {X} p is some element of Cndv - where $X_1 \not\equiv X_2 \Rightarrow \{X_1\}\,p \not\equiv \{X_2\}\,p$ -; for the other cases we put (for R_1,R_2 closed):

$$\{x:=s\}p \equiv p^s_x,\ \{b\}p \equiv (b\supset p), \{\Delta\}p \equiv \underline{\text{false}},\ \{R_1;R_2\}p \equiv \{R_1\}\{R_2\}p,$$
$$\{R_1\cup R_2\}p \equiv \{R_1\}p \wedge \{R_2\}p,\ \{R_1;X\cup R_2\}p \equiv \{R_1\}(\{X\}p) \wedge \{R_2\}p,$$
$$\{\mu X[R_1;X\cup R_2]\} \equiv \nu(\{X\}p)[\{R_1;X\cup R_2\}p]$$

(Note that the last definition has the form of formula (6.1).)

Here p^s_x denotes an extended condition - used for the purpose of this proof only - which has as its meaning $\mathcal{C}(p^s_x)(\varepsilon)(\sigma) = \mathcal{C}(p)(\varepsilon)(\sigma\{^{\mathcal{V}(s)(\sigma)}/x\})$. (Note that the substitution $p[^s/x]$ is defined only for p an assertion, i.e., a first-order formula.) We now first prove that, for all R and p, $\models_c \{R\}p = R\{p\}$, where $\models_c$ denotes validity assuming consistency of the environments, defined here as $\varepsilon(\{X\}p) = \varepsilon(X)\{\mathcal{C}(p)(\varepsilon)\}$. Thus, we show that, for all consistent ε, $\mathcal{C}(\{R\}p)(\varepsilon) = \mathcal{C}(R\{p\})(\varepsilon)$, or, by the definition of $\mathcal{C}$ (definition 6.4b) that (*): $\mathcal{C}(\{R\}p)(\varepsilon) = \mathcal{R}(R)(\varepsilon)\{\mathcal{C}(p)(\varepsilon)\}$. By theorem 4.5 (in its version adapted to greatest fixed points), taking the semantic mapping $\kappa_p(\phi) = \phi\{\mathcal{C}(p)(\varepsilon)\}$ - where κ depends on the parameter p - we have to establish the commutativity result (*) only for R not a μ-term. Verification of (*) for this case is quite standard, and omitted here. This concludes the proof that $\models_c \{R\}p = R\{p\}$. As a consequence, replacing R by $R^\dagger \equiv \mu X[R;X \cup \underline{\text{true}}]$ (with R closed) and dropping the consistency requirement since $R^\dagger$ is closed, we obtain that $\models \{\mu X[R;X \cup \underline{\text{true}}]\}p = R^\dagger\{p\}$. From this it follows that $\models \nu(\{X\}p)[\{R;X \cup \underline{\text{true}}\}p] = R^\dagger\{p\}$, or $\models \nu(\{X\}p)[\{R\}(\{X\}p) \wedge \wedge\, p] = R^\dagger\{p\}$. Taking for $\{X\}p$ its value $Z \in Cndv$, we then obtain $\models \nu Z[\{R\}Z \wedge p] = R^\dagger\{p\}$, and using the equivalence $\models \{R\}Z = R\{Z\}$ then yields the desired result $\models \nu Z[R\{Z\} \wedge p] = R^\dagger\{p\}$. □

By way of conclusion of the paper we briefly discuss two further results of [4] which can be proved using the general strategy from section 4. Both results concern general statements $S \in$ *Stat* (i.e. including general μ-terms). For the first result we extend the definition of *Cond* with constructs $S[p],\ldots$ (rather than $R[p]$). Let the syntactic mapping $\sim$: *Stat* $\to$ *Cond* be defined by $(x{:=}s)^\sim \equiv \tilde{b} \equiv \tilde{\Delta} \equiv \underline{\text{true}}$, $(S_1;S_2)^\sim \equiv \tilde{S}_1 \wedge (S_1\{\tilde{S}_2\})$, $(S_1 \cup S_2)^\sim \equiv \tilde{S}_1 \wedge \tilde{S}_2$, and, as central case

$$(6.2) \qquad \mu X[S]^\sim \equiv \mu\tilde{X}[\tilde{S}[\mu X[S]/X]]$$

where $\tilde{X} \in Cndv$. We show that $\tilde{S}$ is the condition which syntactically expresses that S terminates. This is the content of

<u>THEOREM 6.6</u>. $\models \tilde{S} = S[\underline{\text{true}}]$, *provided the usual consistency condition is satisfied.*

<u>PROOF</u>. Along the same lines as the previous proofs, but now based on a version of theorem 4.5 which starts from the following extension of Lemma 4.3: let $f \in [C \to C]$, $g \in [C \to_s C']$, $h \in C \to_m (C' \to_m C')$. Assume

$$(6.3) \qquad g(f(x)) = h(x)(g(x)).$$

Then $\mu[h(\mu f)]$ exists and $g(\mu f) = \mu[h(\mu f)]$ holds. The general argument of theorem 4.5 - appropriately extended - applies, where κ: $M \to \Pi$ is the semantic mapping yielding, for each $\phi \in M$, the predicate $\kappa(\phi)$ defined by: $\kappa(\phi) = \lambda\sigma.(\bot \notin \phi(\sigma))$. □

The second result concerns a transliteration of the theorem of section 8.3 of [4]. We shall only sketch this case, without developing the full framework necessary for its formulation. Let us consider the following syntactic mapping $\tilde{}$: *Stat* $\to$ (*Cond* $\to$ *Cond*)

$$(x:=s)^{\sim} \equiv \lambda p.p_x^s,\ \tilde{b} \equiv \lambda p.b \supset p,\ \tilde{\Delta} \equiv \lambda p.\ \underline{\text{false}},$$
$$(S_1;S_2)^{\sim} \equiv \tilde{S}_1 \circ \tilde{S}_2, (S_1 \cup S_2)^{\sim} \equiv \tilde{S}_1 \wedge \tilde{S}_2, \mu X[S]^{\sim} \equiv \mu\tilde{X}[\tilde{S}].$$

Here p_x^s is defined as in the proof of theorem 6.5. Let P denote the valuation assigning meaning to $\tilde{S}$ (in $\Pi \to \Pi$) in the natural way. We have

THEOREM 6.7. $P(\tilde{S})(\varepsilon) = \lambda\pi.M(S)(\varepsilon)[\pi]$, *provided the usual consistency condition for* ε *is satisfied.*

PROOF. By the same general argument as used in the preceding proofs. □

As a final remark we mention that we expect the definitions of upper and lower derivative ([9,4]) also to be amenable to a treatment using the general approach of our paper. However, we have not yet found a *semantic* characterization which might be used to justify the syntactic definitions.

REFERENCES

[1] APT, K.R. & G.D. PLOTKIN, *A Cook's tour of countable nondeterminism*, Proc. 8th ICALP (S. Even & O. Kariv, eds), 479-494, Lecture Notes in Computer Science, 115, Springer, 1981.

[2] ARNOLD, A. & M. NIVAT, *Metric interpretations of infinite trees and semantics of nondeterministic recursive programs*, Theoretical Computer Science 11, 181-206, 1980.

[3] BACK, R.J., *Semantics of unbounded nondeterminism*, Proc. 7th ICALP (J.W. de Bakker & J. van Leeuwen, eds), 51-63, Lecture Notes in Computer Science, 85, Springer, 1980.

[4] DE BAKKER, J.W., *Mathematical Theory of Program Correctness*, Prentice-Hall International, 1980.

[5] DE BAKKER, J.W. & W.P. DE ROEVER, *A calculus for recursive program schemes*, Proc. 1st ICALP (M. Nivat, ed.), 167-196, North-Holland, 1973.

[6] BOASSON, L. & M. NIVAT, *Adherences of languages*, J. Comp. Syst. Sciences, 20(3), 281-309, 1980.

[7] DIJKSTRA, E.W., *A discipline of programming*, Prentice-Hall, 1976.

[8] HAREL, D., *First-Order Dynamic Logic*, Lecture Notes in Computer Science, 68, Springer, 1979.

[9] HITCHCOCK, P. & D. PARK, *Induction rules and termination proofs*, Proc. 1st ICALP (M. Nivat, ed.), 225-251, North-Holland, 1973.

[10] KUIPER, R., *An operational semantics for bounded nondeterminism equivalent to a denotational one*, IFIP TC2-MC Symp. on Algorithmic Languages (J.W. de Bakker & J.C. van Vliet, eds), 373-398, North-Holland, 1981.

[11] MILNE, R. & C. STRACHEY, *A Theory of Programming Language Semantics*, 2 Vols., Chapman & Hall, 1976.

[12] NIVAT, M., *Mots infinis engendrés par une grammaire algébrique*, RAIRO Informatique Théorique 11, 311-327, 1977.

[13] NIVAT, M., *Sur les ensembles de mots infinis engendrés par une grammaire algébrique*, RAIRO Informatique Théorique 12, 259-278, 1978.

[14] NIVAT, M., *Infinite words, infinite trees, infinite computations*, Foundations of Computer Science III 2 (J.W. de Bakker & J. van Leeuwen, eds) 3-52, Mathematical Centre Tracts 109, 1979.

[15] PARK, D., *On the semantics of fair parallellism*, *Abstract Software Specification* (D. Bjørner, ed.), 504-526, Lect. Notes in Computer Science, 86, 1980.