Towards a uniform topological treatment of streams and functions on streams

*J.W. de Bakker**
*J.N. Kok***

Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB AMSTERDAM, The Netherlands

ABSTRACT

We study the semantics of functional languages on streams such as Turner's SASL or KRC. The basis of these languages is recursive equations for (functions on) finite or infinite sequences. The paper presents a start towards a mathematical (denotational) description of such languages using tools from metric topology. The description is based on the Banach fixed point theorem and a restricted version of a typed lambda calculus. To a system of recursive stream (function) declarations a system of functions is associated in an appropriate topological domain. These functions have to be contracting in certain arguments and non distance increasing in others; a syntax is designed which ensures the right interplay between these conditions. Nondeterminism is handled by considering compact sets of streams, and preservation of compactness is another important technical issue. Not all concepts in a language such as KRC are covered, and some indications on possible extensions of the framework are provided.

1. Introduction

We present a semantic study of languages with streams and functions on streams as exemplified by Turner's languages SASL and KRC. The tools we use are from metric topology; ultimately, our model relies on Banach's fixed point theorem for contracting functions on a complete metric space.

A *stream* is a finite or infinite sequence of values from a set V. (The examples below always take for V the set of integers.) A *program* is a set of recursive declarations of streams and stream functions, together with an expression to be evaluated with respect to the declarations.

Example: $< v \leftarrow 1 \cdot v_1(v),\ \ v_1 \leftarrow \text{ var } z : (head(z)+1) \cdot v_1(tail(z)) \ \ | \ v >$, we see two declarations, viz. that of the stream v and of the stream function v_1. The expression to be evaluated is v itself. In v_1, the formal z is used which can have a stream (or, in general, a set of streams) as actual. "$\cdot$" denotes concatenation, and "$|$" separates the declarations from the main program expression. The functions *head* and *tail* are as usual. The intended meaning of v is the infinite sequence $1 \cdot 2 \cdot 3 \cdot \cdots$.

Our main task is the development of a semantic framework to assign meaning to declarations of streams and stream functions. First, we define various *metrics*. The distance d between two streams is smaller if the elements where they exhibit their first difference occurs further to the right in the streams. For example, $d(23,24) = \frac{1}{2}$, $d(123,124) = \frac{1}{4}$. By standard topological methods, we extend this metric to sets of streams and to stream functions. Section 2, on topological preliminaries, collects these definitions. Also, the important notions of contracting, non distance increasing, and continuous functions are introduced, and various properties of compactness needed below are described. In fact, compactness, as a limit case of finiteness, is the topological counterpart of the familiar notion of bounded nondeterminism present in various order theoretic approaches.

Section 3 presents the definitions of the syntax and the semantics of our language. The syntax is designed in such a way that the associated semantic functions have the right contracting c.q. non distance increasing properties, as developed in section 4. Two main themes arise here: in a declaration such as $(v \leftarrow \cdots v \cdots)$, recursive occurrences of v have to be *guarded* by some expression (e.g. $v \leftarrow \cdots s \cdot v \cdots$) in order to ensure contractivity. Moreover, in order to guarantee that such contractivity is preserved throughout, stream functions of the type $(v_1 \leftarrow \text{var } z : \cdots z \cdots)$ have to be non distance increasing in z. Appropriate syntactic categories are introduced in order to enforce the right combination of these properties. The format of the syntax follows the usual pattern of a typed lambda calculus, restricted, however, to ground types and first order functional types. We envisage no problems in generalizing this aspect of the syntax.

* Supported in part by ESPRIT project 415: Parallel Architectures and Languages
** Supported by the Netherlands Organisation for the Advancement of Pure Research (Z.W.O.), grant 125-20-04.

The main theorem of the paper is in section 4 where the basic contractivity result, for functions associated with a set of declarations, is established. The intended meaning of the declared streams or stream functions as unique fixed points by Banach's theorem is then immediate. A minor issue to be faced is the possibility that a "guarding" term s in $s.v$ has the empty word in the set $[\![s]\!]\gamma$ denoting its meaning.

Section 5 treats a program as a pair consisting of a set of declarations and an expression. In the latter, we can be more liberal as to the allowed functions occurring in it, since it has only calls (and no declarations) of recursive objects.

Section 6 discusses some limitations and possible extensions. First we discuss functions which, instead of being contracting or non distance increasing, allow a *bounded* increase in distance. The problems which arise in this case are related to those studied by Wadge [22]. Secondly, with a more refined syntax we can also cater for the use of external (i.e. not programmer declared) functions which are not required to be contracting or non distance increasing. However, we then must restrict the way in which such functions occur in our expressions. Thirdly, we mention an important case of a function declaration which is allowed in KRC but does not fit into the present framework (a "permutation" function). We expect that the use of Painlevé limits (rather than of Cauchy sequences of compact sets with respect to the Hausdorff distance) will be useful here, but we have not worked out this idea.

Functional programming in general, and programming on streams in particular, have received wide attention in recent years. For the general background we refer, e.g., to [9] and references contained therein. It will be clear from the above that the languages SASL and KRC [20,21] have been a source of inspiration to us. Further references concerned with programming on streams are [8,10,12,15,23].

Our use of topological techniques goes back to the work of Nivat and his coworkers (e.g. [1,16]). Many of the technical results we use below are also described or applied in [4,5].

An order-theoretic approach to stream semantics is also possible; a basic reference is Broy [6], see also [7] for a more introductory presentation. Advantages of the metric approach are that certain distinctions can be made, in particular between contracting, non distance increasing and (only) continuous functions, which have no direct order-theoretic counterpart. Also, contractivity leads to the attractive situation that uniqueness of fixed points is ensured. However, further work is needed to cope with the problem of unguarded recursion in a topological setting. For connections between metric, order-theoretic, algebraic approaches in general see [17], [18], [19].

2. Topological preliminaries

Let M, M_1, M_2 be metric spaces with (bounded) distances d, d_1, d_2. A function $\phi : M_1 \to M_2$ is called *continuous* whenever for each sequence $\{x_i\}_i$ in M_1 and $x \in M_1$ such that $x = \lim_{i\to\infty} x_i$, we have $\phi(x) = \lim_{i\to\infty}\phi(x_i)$, ϕ is called *contractive* whenever, for each $x,y \in M_1$, $d(\phi(x),\phi(y)) \leqslant c \cdot d(x,y)$ for some constant c with $0 \leqslant c < 1$, and ϕ is called *non distance increasing* whenever for each $x,y \in M_1$, $d(\phi(x),\phi(y)) \leqslant d(x,y)$.

Given a metric space (M,d), d is said to be an ultrametric on M if it satisfies the 'strong triangle inequality': for all $x,y,z \in M$ $d(x,z) \leqslant \max\{ d(x,y),d(y,z) \}$.

Let (M,d) be a complete metric space. For $X,Y \subseteq M$ we define the so called Hausdorff distance

$$\hat{d}(X,Y) = \max(\sup_{x\in X} d'(x,Y), \sup_{y\in Y} d'(y,X)) \text{ with } d'(x,Y) = \inf_{y\in Y} d(x,y).$$

By convention $inf\ \varnothing = 1$ and $sup\ \varnothing = 0$. We now define some spaces obtained from other spaces.

Let $P_{comp}(M)$ denote the non empty compact subsets of M, let $[M_1 \to M_2]$ denote the non distance increasing functions from $M_1 \to M_2$, and let $M_1 \times \cdots \times M_n$ be the Cartesian product of $M_1, \cdots, M_n$. We give these spaces the following metrics:

- $(P_{comp}(M),\hat{d})$: Hausdorff metric induced by the metric on M,
- $([M_1 \to M_2],e)$: $e(\phi_1,\phi_2) = \sup_{x\in M_1} d_2(\phi_1(x),\phi_2(x))$,
- $(M_1 \times \cdots \times M_n,d)$: $d(\langle x_1,\cdots,x_n\rangle, \langle \bar{x}_1,\cdots,\bar{x}_n\rangle) = \max_{i\in\{1,\cdots,n\}} d_i(x_i,\bar{x}_i)$.

2.1. Theorem. *If M, $M_1,\cdots,M_n$ are complete metric spaces then the following spaces with the above defined metrics are also complete: $P_{comp}(M)$, $[M_1 \to M_2]$, $M_1 \times \cdots \times M_n$.*

(i)If $\{X_i\}_i$ is a Cauchy sequence of compact sets in $(P_{comp}(M),\hat{d})$ then there exists a limit and this limit is compact. For details see [4].
(ii) Let $\{\phi_i\}_i$ be a Cauchy sequence in $[M_1 \to M_2]$. Define $\phi' : M_1 \to M_2$ by $\phi'(X) = \lim_i \phi_i(X)$. Then we have $\lim_i \phi_i = \phi'$ and $\phi' \in [M_1 \to M_2]$.
(iii) omitted □.

2.2. Lemma. *If $d,d_1,\cdots,d_n$ are ultrametrics then the above defined metrics are also ultrametrics.*

A well known classical result is Banach's fixed point theorem:

2.3. Theorem. *Let M be d-complete and let $T : M \to M$ be d-contractive. Then T is continuous and has exactly one fixed point X, satisfying $X = \lim_i T^i(X_0)$ for any $X_0 \in M$.*

The following theorem is due to Michael (see [14]). It says that if we 'flatten' compact sets in a certain way we have again a compact set.

2.4. Theorem. *Let X_i, $i \in I$, be compact subsets of M , and let $\{X_i \mid i \in I\}$ be compact in $(P_{comp}(M),\hat{d})$ then $X := \cup \{ X_i \mid i \in I \}$ is compact in (M,d).*

Proof: First we need the following definition: for any finite collection of open sets $U_1,\cdots,U_n$ in M,

$[U_1,\cdots,U_n] := \{X \mid X \subseteq \bigcup_{i=1}^{n} U_i$ and for $i=1,\cdots,n$ $X \cap U_i \neq \varnothing \}$.

We have that $[U_1,\cdots,U_n]$ is open in $(P_{comp}(M),\hat{d})$. Let G be an open cover of X. Then for all i $\in$ I, G is an open cover of X_i, so there is a finite subcollection of G, which covers X_i say $U_i = \{U_{i_1},\cdots,U_{i_n}\}$. Assume for $j=1,\cdots,n$ $U_{i_j} \cap X_i \neq \varnothing$ because otherwise we could remove it. Let $[\overline{U_i}] := [U_{i_1},\cdots,U_{i_n}]$ then it is easy to see that $[\overline{U_i}]$ is an open neighborhood of X_i so $\{[\overline{U_i}] \mid i \in I\}$ is an open cover of $\{X_i | i \in I\}$. So it has a finite subcover, say $\{[\overline{U_{i_1}}],\cdots,[\overline{U_{i_m}}]\}$. So X is covered by the finite collection $\{U_{i_1 1},\cdots,U_{i_1 n_{i_1}}, \quad U_{i_2 1},\cdots,U_{i_2 n_{i_2}}, \quad \cdots \quad U_{i_m 1},\cdots,U_{i_m n_{i_m}}\}$ $\square$.

We list two properties of compact sets which can be found in any standard work on topology, for example [11]: (i) the continuous image of a compact set is compact, (ii) let $X \subseteq M$ and $Y \subseteq N$, X and Y compact then $X \times Y$ is compact in the product topology for $M \times N$.

2.5. Lemma. *Let (M_1,d_1) be a metric space and (M_2,d_2) a metric space where d_2 is an ultrametric. Let $([M_1 \to M_2],e)$ be the metric space of non distance increasing functions from $M_1 \to M_2$ with the bounded function metric. Then we have the following inequality: $d_2(\phi_1(X_1),\phi_2(X_2)) \leq \max(e(\phi_1,\phi_2),d_1(X_1,X_2))$.*

Proof: $d_2(\phi_1(X_1),\phi_2(X_2)) \leq \max(\, d_2(\phi_1(X_1),\phi_2(X_1)),\, d_2(\phi_2(X_1),\phi_2(X_2)) \leq \max(e(\phi_1,\phi_2),d_1(X_1,X_2))$ $\square$.

2.6. Lemma. *Let $\phi : M \to P_{comp}(M)$. Define $\hat{\phi} : P_{comp}(M) \to P(M)$ by $\hat{\phi}(X) = \cup\{\phi(x) \mid x \in X\}$. Then we have $\hat{\phi}: P_{comp}(M) \to P_{comp}(M)$, if ϕ is continuous then $\hat{\phi}$ is continuous, if ϕ is contractive then $\hat{\phi}$ is contractive, if ϕ is non distance increasing then $\hat{\phi}$ is non distance increasing.*

3. Definitions

First we define the the set of streams, and a metric on it so we have a metric space on which we can apply the results of the preceding paragraph on topological spaces. Then we give the syntax and the semantics of our little language.

3.1. Streams

Let $V = \{0,1,2,\cdots\}$ be the set of integers. Let V^{str} be the set of finite and infinite sequences of members of V : $V^{str} = V^* \cup V^\omega$. V^{str} is called the set of streams. Let x and y members of V^{str}.

A metric on the set of streams is defined as follows: $d(x,y) := 2^{-max\{n \mid x[n]=y[n]\}}$ where x[n] denotes the first n numbers of x, with the conventions that if the length of the stream is smaller than n than x[n] = x and $2^{-\infty} = 0$.

X and Y are typical members of $P_{comp}(V^{str})$. Let $\hat{d}$ be the Hausdorff distance on $P_{comp}(V^{str})$.

3.2. Syntax

Let $v \in Var$, where Var is the set of variables. A set of recursive equations (declarations) in our language will look like $\{v_i \leftarrow S_i\}_i$, where the $v_i \in Var$ are the declared variables that can appear recursively in the expressions $S_i \in Exp$. A member of Exp consists of zero or more formal variables $z \in Fovar$ followed by an expression $s \in Exp_1$. If this s is not preceded by formal variables, the expression S is called a stream expression.

In expressions $s \in Exp_1$ we can use several predefined variables. Let $Pvar$ the class of predefined variables. It is composed of the following subclasses with their typical elements: $g \in Cfvar$: set of contracting function variables, $f \in Nfvar$: set of non distance increasing function variables, $\alpha \in Ifvar$: set of integer function variables, if the arity of α is zero then α is called a constant, $\beta \in Sifvar$: set of stream to integer function variables, so $Pvar = Cfvar \cup Nfvar \cup Ifvar \cup Sifvar$.

Now we come to the main definition of this paragraph:

$S \in Exp, s \in Exp_1, t \in Exp_2, a \in Sexp_1, b \in Sexp_2$:

$S ::= \text{var } z_1, \cdots, z_n : s$

$s ::= a \mid z \mid S(s_1) \cdots (s_n) \mid f(s_1) \cdots (s_n) \mid g(t_1) \cdots (t_n) \mid s \cdot t$
$t ::= b \mid tail(s) \mid S(t_1) \cdots (t_n) \mid v(t_1) \cdots (t_n) \mid f(t_1) \cdots (t_n)$

$a ::= \alpha(a_1) \cdots (a_n) \mid \beta(s_1) \cdots (s_n)$
$b ::= \alpha(b_1) \cdots (b_n) \mid \beta(t_1) \cdots (t_n)$

Remarks

Usually we omit the brackets around the arguments of functions or variables.

If an expression s contains a variable v (for example in $s \equiv s'.t$ where $t \equiv v(t_1) \cdots (t_n)$) then that variable is protected by some other expression s', it is called guarded, or it appears unguarded within the arguments of a contracting function variable, but then we also call it guarded. If a variable is guarded then it is not necessary to guard variables that appear within the arguments of that variable. Exp_2 is a class which contains expressions that have appearances of unguarded variables.

It may seem that if one uses a formal variable as a guard we can have unguarded variables. For example in $v \leftarrow \text{var } z : z \cdot v(tail(z))$ we can get an unguarded v if the formal variable z obtains ϵ (the empty stream) as meaning of its corresponding actual parameter. We will define our semantics of a set of declarations in such a way that in these cases a default value is taken.

The classes $Sexp_1$ and $Sexp_2$ are meant as a classes of integers, the basic building blocks of our streams.

The general idea is that the so constructed expressions s are in a certain sense contractive in variables v and non distance increasing in formal variables z.

Examples of functions f would be "$\cup$" (non deterministic choice) and "||" (merge), as defined in [4]. *Fair* merge, however, does not preserve closedness and does not fit naturally into our framework.

3.3. Types and domains

We will use the domains DOM_ω for sets of streams and $DOM_{\omega^n \to \omega}$ for functions on streams.

Let $DOM_\omega = (P_{comp}(V^{str}), \hat{d})$ be the non empty compact subsets of V^{str} with the Hausdorff metric. Let $DOM_{\omega^n \to \omega} = ([P_{comp}(V^{str}) \times \cdots \times P_{comp}(V^{str}) \to P_{comp}(V^{str})], e)$ be the set of non distance increasing functions from $(P_{comp}(V^{str}) \times \cdots \times P_{comp}(V^{str}), d_{\max})$ to $(P_{comp}(V^{str}), \hat{d})$ equipped with the bounded function metric e. Let $DOM = DOM_\omega \cup DOM_{\omega^* \to \omega}$.

Let Env be the class of environments. $\gamma \in Env$ is a total function from $Var \cup Pvar \cup Fovar \to DOM$, which satisfies the following points:

- $f \in Nfvar$ implies $\gamma(f) \in DOM_{\omega^n \to \omega}$ and $\gamma(f)$ is monotone w.r.t. set inclusion,
- $g \in Cfvar$ implies $\gamma(g) \in DOM_{\omega^n \to \omega}$, and moreover, $\gamma(g)$ is contractive. We also require that $\gamma(g)$ is monotone (w.r.t. set inclusion),
- $\alpha \in Ifvar$ implies $\gamma(\alpha) \in P_{finite}(V) \times \cdots P_{finite}(V) \to P_{finite}(V)$,
- $\beta \in Sifvar$ implies $\gamma(\beta) \in P_{comp}(V^{str}) \times \cdots P_{comp}(V^{str}) \to P_{finite}(V)$ and $\gamma(\beta)$ is non distance increasing,
- $z \in Fovar$ implies $\gamma(z) \in DOM_\omega$,
- $v \in Var$ implies $\gamma(v) \in DOM_\omega$ or $\gamma(v) \in DOM_{\omega^* \to \omega}$.

Remarks

(i)$\gamma(\beta) \in DOM_{\omega^* \to \omega}$ because $P_{finite}(V) \subset P_{comp}(V^{str})$, and for $\gamma(\alpha)$ the requirement of being non distance increasing is trivially satisfied.

(ii)The condition that $\gamma(\beta)$ is non distance increasing implies that this function is determined by the first elements of its arguments. At first sight this is very restrictive. However, the arguments can sometimes depend on more elements, and, furthermore, the syntax can be extended by allowing (at some places), functions that are determined by more elements.

(iii)In case a functionvariable f or g is used in an expression s we also require for all $X \in P_{comp}(V^{str})$ $\epsilon \notin \gamma(f)(X)$ and $\epsilon \notin \gamma(g)(X)$.

Type is the set of types. A type τ is either ground or functional. Ground types are ω_0 and ω and functional types are

$$\omega \to \omega, \omega \times \omega \to \omega, \cdots, \omega^n \to \omega, \cdots, \omega_0 \to \omega_0, \omega_0 \times \omega_0 \to \omega_0, \cdots, \omega_0^n \to \omega_0, \cdots, \omega \to \omega_0, \cdots, \omega^n \to \omega_0, \cdots.$$

We consider only expressions that can be typed with typing rules. For example: $\alpha \in Ifvar : type(\alpha) = \omega_0^n \to \omega_0$, $v \in Var : type(v) = \omega$ or $type(v) = \omega^n \to \omega$.

Now we give maps that define the semantics, in the sequel we will prove that the domain of these maps is DOM. The semantics are defined by the following maps:

Let $[\![\cdots]\!] : Exp \to Env \to DOM$ be defined by

$$[\![\text{var } z_1, \cdots, z_n : s]\!]\gamma = \lambda\phi_1. \cdots \lambda\phi_n . [\![s]\!]_1 \gamma \{\phi_i / z_i\}_{i=1}^n$$

where $\bar{s} \equiv$ if $z_1 = \epsilon$ or $\cdots$ or $z_n = \epsilon$ then ϵ else s fi. Remark: the replacement of the s by the if..fi construct is done for two reasons. First we do not want variables to become unguarded. Consider the following set of declarations, where we define a function "v" and a stream "$\bar{v}$": { $v \leftarrow \text{var } z : z \cdot v(tail(z))$, $\bar{v} \leftarrow 1 \cdot v(1)$ }. Now if "v" obtains ϵ as actual parameter, then it is not clear what this definition means, because the variable v in the body of $v \leftarrow \text{var } z : z \cdot v(tail(z))$ becomes unguarded.

Second we want $[\![S]\!]\gamma$ for all $\gamma \in Env$ to be a non distance increasing function. Without the if $\cdots$ fi construct this is not always the case. For example, let $S \equiv \text{var } z_1, z_2 : z_1 \cdot tail(z_2)$, $\gamma \in Env$, so $[\![\text{var } z_1, z_2 : z_1 \cdot tail(z_2)]\!]\gamma = \lambda\phi_1.\lambda\phi_2.(\phi_1 \cdot tail(\phi_2))$. Now if $[\![S]\!]\gamma$ would be a non distance increasing function, we would have $d([\![S]\!]\gamma(\epsilon)(123), [\![S]\!]\gamma(\epsilon)(1234)) \leq d(<\epsilon,123>, <\epsilon,1234>)$ but we have instead $d([\![S]\!]\gamma(\epsilon)(123), [\![S]\!]\gamma(\epsilon)(1234)) = 2.d(<\epsilon,123>, <\epsilon,1234>)$ so $[\![S]\!]\gamma$ is not non distance increasing.

Let $[\![\cdots]\!]_1 : Exp_1 \rightarrow Env \rightarrow DOM$ be defined by

$$
\begin{aligned}
&[\![a]\!]_1\ \gamma = [\![a]\!]_a\ \gamma,\\
&[\![s \cdot t]\!]_1\ \gamma = [\![s]\!]_1\ \gamma \cdot [\![t]\!]_2\ \gamma,\\
&[\![S(s_1) \cdots (s_n)]\!]_1\ \gamma = [\![S]\!]\ \gamma\ \ [\![s_1]\!]_1\ \gamma \cdots [\![s_n]\!]_1\ \gamma,\\
&[\![f(s_1) \cdots (s_n)]\!]_1\ \gamma = \gamma(f)\ [\![s_1]\!]_1\ \gamma \cdots [\![s_n]\!]_1\ \gamma,\\
&[\![g(t_1) \cdots (t_n)]\!]_1\ \gamma = \gamma(g)\ [\![t_1]\!]_2\ \gamma \cdots [\![t_n]\!]_2\ \gamma.
\end{aligned}
$$

Let $[\![\cdots]\!]_2 : Exp_2 \rightarrow Env \rightarrow DOM$ be defined by

$$
\begin{aligned}
&[\![b]\!]_2\ \gamma = [\![b]\!]_b\ \gamma,\\
&[\![v(t_1) \cdots (t_n)]\!]_2\ \gamma = \gamma(v)\ \ [\![t_1]\!]_2\ \gamma \cdots [\![t_n]\!]_2\ \gamma,\\
&[\![S(t_1) \cdots (t_n)]\!]_1\ \gamma = [\![S]\!]\ \gamma\ \ [\![t_1]\!]_2\ \gamma \cdots [\![t_n]\!]_2\ \gamma,\\
&[\![f(t_1) \cdots (t_n)]\!]_1\ \gamma = \gamma(f)\ [\![t_1]\!]_2\ \gamma \cdots [\![t_n]\!]_2\ \gamma.
\end{aligned}
$$

Let $[\![\cdots]\!]_a : Sexp_1 \rightarrow Env \rightarrow DOM$ be defined by

$$
\begin{aligned}
&[\![\alpha(a_1) \cdots (a_n)]\!]_a\ \gamma = \gamma(\alpha)\ [\![a_1]\!]_a\ \gamma \cdots [\![a_n]\!]_a\ \gamma,\\
&[\![\beta(s_1) \cdots (s_n)]\!]_a\ \gamma = \gamma(\beta)\ [\![s_1]\!]_1\ \gamma \cdots [\![s_n]\!]_1\ \gamma.
\end{aligned}
$$

Let $[\![\cdots]\!]_b : Sexp_2 \rightarrow Env \rightarrow DOM$ be defined by

$$
\begin{aligned}
&[\![\alpha(b_1) \cdots (b_n)]\!]_b\ \gamma = \gamma(\alpha)\ [\![b_1]\!]_b\ \gamma \cdots [\![b_n]\!]_b\ \gamma,\\
&[\![\beta(t_1) \cdots (t_n)]\!]_b\ \gamma = \gamma(\beta)\ [\![t_1]\!]_2\ \gamma \cdots [\![t_n]\!]_2\ \gamma.
\end{aligned}
$$

We have $type(S) \in \{\omega, \omega \rightarrow \omega, \cdots, \omega^n \rightarrow \omega, \cdots\}$. Recall that we only consider expressions that can be typed, we usually omit the subscripts of the $[\![\cdots]\!]$ functions.

3.3.1. Theorem. *If an expression S has type τ then we have $[\![S]\!]\gamma \in DOM_\tau$.*

Proof: We prove that for $S \equiv \text{var } z_1, \cdots, z_n : s$ we have $[\![S]\!]\gamma \in P_{comp}(V^{str}) \times \cdots \times P_{comp}(V^{str}) \rightarrow P_{comp}(V^{str})$.

We have for all $s \in Exp_1$, $n \in \{0,1, \cdots\}$, $z_1, \cdots, z_n \in Fovar$, $\gamma \in Env$

$$[\![\text{var } z_1, \cdots, z_n : s]\!]\gamma \in P_{comp}(V^{str}) \times \cdots \times (V^{str}) \rightarrow P_{comp}(V^{str}) \text{ iff } [\![s]\!]\gamma \in P_{comp}(V^{str}) \qquad (*)$$

So we have to prove for all $s \in Exp_1$, $\gamma \in Env$: $[\![s]\!]\gamma \in P_{comp}(V^{str})$. This is proved simultaneously with $[\![t]\!]\gamma \in P_{comp}(V^{str})$, $[\![a]\!]\gamma \in P_{finite}(V)$, $[\![b]\!]\gamma \in P_{finite}(V)$. Induction on the complexity of s,t,a,b. We do not treat all cases.

$s \equiv a$: Either $a \equiv \alpha(a_1) \cdots (a_n)$ or $a \equiv \beta(s_1) \cdots (s_n)$. We treat $a \equiv \alpha(a_1) \cdots (a_n)$. By induction $[\![a_i]\!]\gamma \in P_{finite}(V)$, $i = 1, \cdots, n$. So $<[\![a_1]\!]\gamma, \cdots, [\![a_n]\!]\gamma> \in P_{finite}(V) \times \cdots \times P_{finite}(V)$.
We have $\gamma(\alpha) \in P_{finite}(V) \times \cdots \times P_{finite}(V) \rightarrow P_{finite}(V)$. Observe $P_{finite}(V) \subset P_{comp}(V^{str})$.

$s \equiv f(s_1) \cdots (s_n)$: By induction $[\![s_i]\!]\gamma \in P_{comp}(V^{str})$, so $<[\![s_1]\!]\gamma, \cdots, [\![s_n]\!]\gamma> \in P_{comp}(V^{str}) \times \cdots \times P_{comp}(V^{str})$.
We have $\gamma(f) \in P_{comp}(V^{str}) \times \cdots \times P_{comp}(V^{str}) \rightarrow P_{comp}(V^{str})$.

$s \equiv s \cdot t$: $[\![s]\!]\gamma, [\![t]\!]\gamma \in P_{comp}(V^{str})$ by induction, so $[\![s]\!]\gamma \times [\![t]\!]\gamma = \{<x,y> \mid x \in [\![s]\!]\gamma, y \in [\![t]\!]\gamma\}$
is compact in $V^{str} \times V^{str}$. We have that "." is continuous from $V^{str} \times V^{str}$ so the result follows.

$t \equiv S(t_1) \cdots (t_n)$: $S \equiv \text{var } z_1, \cdots, z_n : s$ and by induction we know $[\![s]\!]\gamma \in P_{comp}(V^{str})$, so by (*) we have
$[\![S]\!]\gamma = [\![\text{var } z_1 \cdots z_n : s]\!]\gamma \in P_{comp}(V^{str}) \times \cdots \times P_{comp}(V^{str}) \rightarrow P_{comp}(V^{str})$.

$a \equiv \alpha(a_1) \cdots (a_n)$: By induction $<[\![a_1]\!]\gamma, \cdots, [\![a_n]\!]\gamma> \in P_{finite}(V) \times \cdots \times P_{finite}(V)$
so $\gamma(\alpha) < [\![a_1]\!]\gamma, \cdots, [\![a_n]\!]\gamma > \in P_{finite}(V)$.

The fact that $[\![S]\!]\gamma$ is non distance increasing will be proved in the main theorem of next section and so we are done □.

4. Declarations

The syntax and the semantics of a set of declarations will be given in this section.

4.1. Syntax

Let d $\in$ Decl be a set of declarations: $d ::= \{v_1 \leftarrow S_1, \cdots, v_n \leftarrow S_n\}$ where $v_i \in Var, S_i \in Exp, i = 1, \cdots, n$. We consider only sets of declarations d such that

- d can be typed,
- for each variable v occurring in some S_j we have that $v \in \{v_1, \cdots, v_n\}$ (this is called closedness of d),
- all z occurring in S_j are in the scope of some var z, and hence, we have no z's occurring in a global way.

4.2. Semantics

To define the semantics we associate with a set of declarations $d \equiv \{v_i \leftarrow S_i\}_i$ a function F:

$$F : DOM_{\tau_1} \times \cdots \times DOM_{\tau_n} \rightarrow DOM_{\tau_1} \times \cdots \times DOM_{\tau_n},$$

$$F = \lambda\phi_1^{\tau_1}. \cdots . \lambda\phi_n^{\tau_n}. < [\![S_1]\!]\gamma\{\phi_i / v_i\}_{i=1}^n, \cdots, [\![S_n]\!]\gamma\{\phi_i / v_i\}_{i=1}^n >$$

where $\gamma \in Env$, $\text{type}(v_i) = \tau_i$ for i=1,...,n. We will prove that F is contractive on $DOM_{\tau_1} \times \cdots \times DOM_{\tau_n}$ so there exists a unique fixed point of F. This fixed point will be taken as the meaning of a set of declarations.

4.2.1. Lemma. *If for all $S \in Exp$, $v \in Var$, $\gamma \in Env$ $\lambda\phi.[\![S]\!]\gamma\{\phi / v\}$ is contractive, then for all $d \in Decl$ its associated function F is contractive for all $\gamma \in Env$.*

Let $Env^* = \{\gamma \in Env \mid \text{for all } z \in Fovar : \epsilon \notin \gamma(z)\}$.

4.2.2. Lemma. *Let $s \in Exp_1$, and let $z_1, \cdots, z_n \in Fovar$ the free formal variables of s. Then we have $\lambda\phi.[\![\text{if } z_1 = \epsilon \text{ or } \cdots \text{ or } z_n = \epsilon \text{ then } \epsilon \text{ else } s \text{ fi}]\!]\gamma\{\phi / v\}$ is contractive for all $\gamma \in Env$ iff $\lambda\phi.[\![s]\!]\gamma\{\phi / v\}$ is contractive for all $\gamma \in Env^*$ and $\lambda\phi.[\![\text{if } z_1 = \epsilon \text{ or } \cdots \text{ or } z_n = \epsilon \text{ then } \epsilon \text{ else } s \text{ fi}]\!]\gamma\{\phi / z\}$ is non distance increasing for all $\gamma \in Env$ iff $\lambda\phi.[\![s]\!]\gamma\{\phi / z\}$ is contractive for all $\gamma \in Env^*$ and for all ϕ such that $\epsilon \notin \phi$.*

4.2.3. Lemma. *If $\epsilon \notin \phi_i, i = 1, \cdots, n$ then $\epsilon \notin [\![\text{var } z_1, \cdots, z_n : s]\!]\gamma(\phi_1) \cdots (\phi_n)$ for all $\gamma \in Env$.*

4.2.4. Lemma. *Let $\gamma \in Env^*$. Let $length(x)$ denote the length of a stream x and $length(X) = inf\{length(x) \mid x \in X\}$. Then for all $s \in Exp_1$ $length([\![s]\!]\gamma) \geqslant 1$.*

4.2.5. Theorem. *Let $d \in Decl$. Then F (associated with d) is contractive on $DOM_{\tau_1} \times \cdots \times DOM_{\tau_n}$.*

Proof. By (4.2.1) and (4.2.2) it suffices to prove for all $\gamma \in Env^*$, $v \in Var$, $s \in Exp_1$

$$d([\![s]\!]\gamma\{\phi / v\}, [\![s]\!]\gamma\{\phi' / v\}) \leqslant c.d(\phi,\phi'), 0 \leqslant c < 1.$$

This is proved simultaneously with the following facts. For all $v \in Var$, $z \in Fovar$, $s \in Exp_1$, $t \in Exp_2$, $a \in Sexp_1$, $b \in Sexp_2$, $\gamma \in Env^*$

$$d([\![s]\!]\gamma\{\phi / z\}, [\![s]\!]\gamma\{\phi' / z\}) \leqslant d(\phi,\phi') \text{ for all } \phi,\phi' \text{ such that } \epsilon \notin \phi,\phi',$$

$$d([\![t]\!]\gamma\{\phi / v\}, [\![t]\!]\gamma\{\phi' / v\}) \leqslant d(\phi,\phi') \text{ for all } \phi,\phi',$$

$$d([\![t]\!]\gamma\{\phi / z\}, [\![t]\!]\gamma\{\phi' / z\}) \leqslant 2.d(\phi,\phi') \text{ for all } \phi,\phi' \text{ such that } \epsilon \notin \phi,\phi',$$

$$d([\![a]\!]\gamma\{\phi / v\}, [\![a]\!]\gamma\{\phi' / v\}) \leqslant c.d(\phi,\phi') \text{ for all } \phi,\phi',$$

$$d([\![a]\!]\gamma\{\phi / z\}, [\![a]\!]\gamma\{\phi' / z\}) \leqslant d(\phi,\phi') \text{ for all } \phi,\phi' \text{ such that } \epsilon \notin \phi,\phi',$$

$$d([\![b]\!]\gamma\{\phi / v\}, [\![b]\!]\gamma\{\phi' / v\}) \leqslant d(\phi,\phi') \text{ for all } \phi,\phi',$$

$$d([\![b]\!]\gamma\{\phi / z\}, [\![b]\!]\gamma\{\phi' / z\}) \leqslant 2.d(\phi,\phi') \text{ for all } \phi,\phi' \text{ such that } \epsilon \notin \phi,\phi'.$$

The proof goes by induction on the (structural) complexity of s,t,a,b.

$s \equiv a$: Either $a \equiv \alpha(a_1) \cdots (a_n)$ or $\beta(s_1) \cdots (s_n)$. We treat $\beta(s_1) \cdots (s_n)$.

By induction for $i = 1, \cdots, n$ $d([\![s_i]\!]\gamma\{\phi / v\}, [\![s_i]\!]\gamma\{\phi' / v\}) \leqslant c.d(\phi,\phi')$

so $d(<[\![s_1]\!]\gamma\{\phi / v\}, \cdots, [\![s_n]\!]\gamma\{\phi / v\}>, <[\![s_1]\!]\gamma\{\phi' / v\}, \cdots, [\![s_n]\!]\gamma\{\phi' / v\}>) \leqslant c.d(\phi,\phi')$.

$\gamma(\beta)$ is non distance increasing, so we are done. The result for $z \in Fovar$ follows by analogy.

$s \equiv z$: $d([\![z]\!]\gamma\{\phi / v\}, [\![z]\!]\gamma\{\phi' / v\}) = 0 \leqslant c.d(\phi,\phi')$, $d([\![z]\!]\gamma\{\phi / z\}, [\![z]\!]\gamma\{\phi' / z\}) \leqslant d(\phi,\phi')$.

$s \equiv f(s_1) \cdots (s_n)$: By induction for $i = 1, \cdots, n$ $d([\![s_i]\!]\gamma\{\phi / v\}, [\![s_i]\!]\gamma\{\phi' / v\}) \leqslant c.d(\phi,\phi')$, so $d(<[\![s_1]\!]\gamma\{\phi / v\}, \cdots, [\![s_n]\!]\gamma\{\phi / v\}>, <[\![s_1]\!]\gamma\{\phi' / v\}, \cdots, [\![s_n]\!]\gamma\{\phi' / v\}>) \leqslant c.d(\phi,\phi')$.

$\gamma(f)$ is non distance increasing, so we are done. The result for $z \in Fovar$ follows by analogy.

$s \equiv s{\cdot}t$: By induction $d([\![s]\!]\gamma\{\phi/v\},[\![s]\!]\gamma\{\phi'/v\}) \leqslant c.d(\phi,\phi')$ and $d([\![t]\!]\gamma\{\phi/v\},[\![t]\!]\gamma\{\phi'/v\}) = d(\phi,\phi')$

By properties of concatenation we are done if $d([\![s]\!]\gamma\{\phi/v\},[\![s]\!]\gamma\{\phi'/v\}) > 0$.

So suppose $[\![s]\!]\gamma\{\phi/v\} = [\![s]\!]\gamma\{\phi'/v\}$. $\gamma\{\phi/v\} \in Env^*$, so by lemma (4.2.4) $length([\![s]\!]\gamma\{\phi/v\}) \geqslant 1$ so

$$d([\![s{\cdot}t]\!]\gamma\{\phi/v\},[\![s{\cdot}t]\!]\gamma\{\phi'/v\}) \leqslant \tfrac{1}{2}\quad d([\![t]\!]\gamma\{\phi/v\},[\![t]\!]\gamma\{\phi'/v\})$$

For $z \in Fovar$ we have again (because $\epsilon \notin \phi$) $\gamma\{\phi/v\} \in Env^*$.

$t \equiv v'(t_1)\cdots(t_n)$: We have:

$$d(<[\![t_1]\!]\gamma\{\phi/v\},\cdots,[\![t_n]\!]\gamma\{\phi/v\}>,<[\![t_1]\!]\gamma\{\phi'/v\},\cdots,[\![t_n]\!]\gamma\{\phi'/v\}>) \leqslant d(\phi,\phi').$$

Now if $v \not\equiv v'$ then we have $\gamma(v')$ is non distance increasing and we are done and if $v \equiv v'$ we apply lemma (2.5) and remark that ϕ and ϕ' are non distance increasing.

$t \equiv S(t_1)\cdots(t_n)$: Let $S \equiv \text{var } z_1,\cdots,z_n : s$. By induction we have

$d([\![s]\!]\gamma\{\phi/z\},[\![s]\!]\gamma\{\phi'/z\}) \leqslant d(\phi,\phi')$ for all $\gamma \in Env^*$ and ϕ,ϕ' such that $\epsilon \notin \phi,\phi'$. So by lemma (4.2.2) $[\![S]\!]\gamma$ is non distance increasing for all γ. Now if $[\![S]\!]\gamma\{\phi/v\} \neq [\![S]\!]\gamma\{\phi'/v\}$ we apply lemma (2.5).

Other cases are omitted □.

$[\![\cdots]\!]_{Decl}$: $Decl \to DOM_{\tau_1} \times \cdots \times DOM_{\tau_n}$ is defined by $[\![\{v_i^{\tau_i} \leftarrow S_i\}_{i=1}^n]\!]_{Decl} = Fix(F)$,

the fixed point of F which exists by the Banach theorem. This fixed point can be obtained by iterating from an arbitrary starting point in the space.

5. Programs

5.1. Syntax

A program is a a set of declarations combined with an expression u. This expression u is a member of $Eexp$, a class of expressions. $Eexp$ leaves in a certain sense more freedom than Exp_1 and Exp_2. We do not have to restrict our class of functions. Let δ be a new member of $Fvar$. We do not put any new restrictions on $\gamma \in Env$, so we only know that $\gamma(\delta)$ is a function.

$$u \in Eexp \quad u ::= a \mid v(u_1)\cdots(u_n) \mid \delta(u_1)\cdots(u_n)$$

Let $p \in Prog$ be a program. $Prog$ is defined by $p ::= <d \mid u>$ where $d \in Decl$ and $u \in Eexp$. Assume furthermore that every $p \in Prog$ is syntactically closed, i.e. every $v \in Var$ that appears in u is declared in d. Recall that d itself is closed.

5.2. Semantics

$[\![\cdots]\!]_E : Eexp \to Env \to P(V^{str})$ is defined by

$$[\![a]\!]_E\gamma = [\![a]\!]\gamma,$$
$$[\![v(u_1)\cdots(u_n)]\!]_E\gamma = \gamma(v)([\![u_1]\!]_E\gamma\cdots[\![u_n]\!]_E\gamma,$$
$$[\![\delta(u_1)\cdots(u_n)]\!]_E\gamma = \gamma(\delta)([\![u_1]\!]_E\gamma\cdots[\![u_n]\!]_E\gamma.$$

$[\![\cdots]\!]_{Prog} : Prog \to Env \to P(V^{str})$ is defined by

$$[\![<d \mid u>]\!]_{Prog}\gamma = [\![u]\!]_E\gamma\{\epsilon_i([\![d]\!]_{Decl})/v_i\}_i$$
where $\epsilon_i(\cdots)$ denotes the i^{th} component of $\cdots$.

6. Some extensions and remarks

6.1. Extensions

In this section we want to discuss some extensions and we will see some limitations of our framework. The treatment of these ideas will be brief and not fully worked out. First we look at three possible extensions:

- Use a fully typed lambda calculus. We think there are no new problems when we allow functions of higher order than we have used up to now.

- Allow more declarations: There are sets of declarations that are intuitively correct and are not allowed in our framework. For example $\{v_1 \leftarrow v_2, v_2 \leftarrow 1.(v_2+1)\}$ does not satisfy our syntax. We like to allow such a set of declarations. One way to do this is the derivation of an equivalent correct set of declarations. We can syntactically transform such a set with certain rules. An example of such a rule would be:

  Let $v \in Var$ be a variable that is declared in a set of declarations d, i.e. $d \equiv \{\cdots, v \leftarrow S, \cdots\}$. Let this variable v also appear in the body of a declaration: there is a $\bar{v} \in Var$ such that $d \equiv \{\cdots, \bar{v} \leftarrow \cdots v \cdots, \cdots\}$. Now replace this occurrence of v by its body S.

Note that clashes of formals do not occur due to the absence of global variables. Our example $\{v_1 \leftarrow v_2, v_2 \leftarrow 1.(v_2+1)\}$ could be transformed to $\{v_1 \leftarrow 1.(v_2+1), v_2 \leftarrow 1.(v_2+1)\}$. The latter set of declarations is permitted in our framework.

• In a set of declarations we can define functions. Up to now these functions were all non distance increasing. We can extend our sets of declarations in such a way that it is possible to declare other kinds of functions.

(i) functions from $P_{comp}(V^{str}) \to P_{comp}(V^{str})$ that satisfy for a fixed n $d(\phi(x),\phi(y)) \leqslant 2^n . d(x,y)$.

We have to make expressions u such that for all $v \in Var$, $z \in Fovar$ $d([\![u]\!]\gamma\{\phi / v\},[\![u]\!]\gamma\{\bar{\phi} / v\}) \leqslant c.d(\phi,\bar{\phi})$, $d([\![u]\!]\gamma\{\phi / z\},[\![u]\!]\gamma\{\bar{\phi} / z\}) \leqslant 2^n . d(\phi,\bar{\phi})$.

If we want to use these functions in declarations of non distance increasing functions then we must be careful. For example, suppose we have constructed a function that can make the distance between two streams twice as big. We want to use this function in the definition of a non distance increasing function. This can only be done in places where we were allowed to use the tail function. For $n \geqslant 2$ relatively minor refinements in the syntactic and semantic framework developed above are necessary. Related questions are discussed by Wadge [22].

(ii) Monotone functions (with respect to set inclusion). We now discuss what happens when we lift the restriction on f and g imposed before. Specifically, we replace f,g by functionsymbols $\delta \in Mofvar$ (Monotone function variables) the meaning of which is required only to be a monotone function from $P_{comp}(V^{str}) \to P_{comp}(V^{str})$. Functions $[\![S]\!]\gamma$ are now no longer non distance increasing, and an additional requirement on the set of declarations is necessary to ensure that $\lambda\phi.[\![S]\!]\gamma\{\phi / v\}$ is contracting.

We define the classes $Mexp, Mexp_1, Mexp_2$. Let $S \in Mexp, s \in Mexp_1, t \in Mexp_2$. We use S, s, t in analogy with our earlier syntax, but they are not the same.

$S ::= \text{var } z_1, \cdots, z_n : s$

$s ::= a \mid z \mid \delta(t_1) \cdots (t_n) \mid S\ (t_1) \cdots (t_n) \mid s.t$

$t ::= a \mid z \mid \delta(t_1) \cdots (t_n) \mid v\ (t_1) \cdots (t_n) \mid S\ (t_1) \cdots (t_n)$

In the definition of the semantics we take the usual precautions to ensure that the guards of a $v \in Var$ can not become ϵ. The formal definition of the semantics is omitted.

Let $d \in Decl$ be a set of declarations, $d = \{v_i \leftarrow S_i\}_i$. We now formulate the restriction on d. First we introduce some terminology. Let $FV(d)$ be the set of formal variables in $\bigcup_{i=1}^{n} S_n$. Let $OF(d) \subset FV(d)$ denote the outer formal variables in d : $OF(d) = \bigcup_{i=1}^{n} \{z_{i_1}, \cdots, z_{i_{n(i)}}\}$ if $S_i \equiv \text{var } z_{i_1}, \cdots, z_{i_{n(i)}} : s_i$. Let $IF(d) = FV(d) - OF(d)$ be the inner formals of d.

Because the set of declarations is well typed, every $z \in IF(d)$ is instantiated with an expression $s \in Exp_1$ or $t \in Exp_2$. Now we want to define what an instantiation means for a $z \in OF(d)$. If $z \in OF(d)$ there is a variable $v \in Var$ such that $(v \leftarrow \text{var} \cdots z \cdots : s) \in d$. Assume that this v is used in the body of a variable $\bar{v} \in Var$, i.e. $(v \leftarrow \text{var} \cdots : \cdots v(t_1) \cdots (t_n) \cdots) \in d$. $\bar{v}$ is not necessarily different from v. Let t_i the expression that appears in the body of $\bar{v}$ as 'actual' expression for z. This t_i is called an instantiation of z.

An expression is *allowed* as argument of a $\delta \in Mofvar$ in a set of declarations $d \in Decl$ if no $v \in Var$ occurs in this expression, and if $z \in Fovar$ occurs in this expression, then all instantiations of z must be allowed.

Now we can formulate the restriction. All arguments of a $\delta \in Mofvar$ that appear in a $d \in Decl$ must be allowed. We omit the proof that this restriction implies the desired contractivity property.

Second, we want to discuss how we can mix monotone functions with sets of declarations of other functions. For simplicity, consider only two kinds of functions: monotone and non distance increasing. Let d be a set of declarations in which we define (simultaneously) these two kinds of functions. There are two kinds of variables: those that will be assigned as meaning a non distance increasing function and those that will be assigned a monotone function. The difference between these two kinds of variables will be denoted by putting a bar on the latter.

Let $S \in Exp$ be expressions that belong to v's and $\bar{S} \in Mexp$ expressions that belong to $\bar{v}$'s, i.e. $d = \{v_i \leftarrow S_i, \bar{v}_i \leftarrow \bar{S}_i\}_i$. We extend Exp and $Mexp$ such that $\bar{v}$ can be used in S and v in $\bar{S}$. In an expression $S \in Exp$ we can use $\bar{v}$ in places where v is allowed, but the arguments of a $\bar{v}$ must be constant expressions. An expression is called constant if it contains no $v \in Var$ or $z \in Fovar$. In an expression $\bar{S} \in Mexp$ v can be used on each place where a $\bar{v}$ is allowed. The conditions together ensure that the v_i are non distance increasing.

In general, monotone functions are not continuous with respect to the Hausdorff distance. Still, we would like to have that finite approximations converge to the result. We illustrate this problem by the permutation function. This function is described by Turner in [20]. It maps a stream to the set of all its permutations. Let x be a stream and $\{x_i\}_i$ a sequence such that $\lim_{n\to\infty} x_i$. The x_i can be considered as approximations of x. We have that a sequence of permutations of successive approximations is not necessarily a converging sequence. Take for example $x = 12345 \cdots$. Let $x[n]$ denote the first n elements of x, i.e. $x[n] = 1 \cdots n$. We have $\lim_{n\to\infty} x[n] = x$, but $\lim_{n\to\infty} perm(x[n])$ does not exist.

A way out could be an extension of the notion of limit we use. Arnold, Nivat [1] consider operators that are only continuous with respect to Painlevé limits. Let Lim denote a limit in the Painlevé sense. Then we have

$\lim_{n\to\infty} perm(x[n]) = perm(x)$. See also some of the remarks of Smyth [18] where he compares the Hausdorff metric with the Vietoris topology. They agree on $P_{comp}(V^{str})$ but differ on $P_{closed}(V^{str})$. These are topics for further research.

6.2. Remarks

Up to now, our functions have as domain the collection of compact sets of streams. Sometimes it is more convenient to take as domain the set of streams, and consider functions from $V^{str} \to P_{comp}(V^{str})$. When we want to apply the function to a set of streams we do this in the usual way, i.e. $f[X] = \cup\{f(x) \mid x \in X\}$. An appeal to lemma 2.6. yields that functions extended in this way fit into our framework.

Lemma 2.6. can also simplify proofs. For example, to prove that a (monotonic) function is non distance increasing from $P_{comp}(V^{str}) \to P_{comp}(V^{str})$ it suffices to show that this function is non distance increasing from $V^{str} \to P_{comp}(V^{str})$.

7. References

[1] ARNOLD, A., NIVAT, M., The metric space of infinite trees, algebraic and topological properties, Fund. Inform. III, 4, pp 445-476, 1980.

[2] DE BAKKER, J.W., Mathematical theory of program correctness, Prentice Hall International,1980.

[3] DE BAKKER, J.W., KLOP, J.W., MEYER, J.-J.CH., Correctness of programs with function procedures in Logic of Programs, LNCS 131, proceedings 1981, Kozen, D., Springer, 1982.

[4] DE BAKKER, J.W., ZUCKER, J.I., Compactness in semantics for merge and fair merge in Logic of Programs, LNCS 164, proceedings 1983, Kozen, D., Clarke, E., Springer, 1983.

[5] DE BAKKER, J.W., ZUCKER, J.I., Processes and the denotational semantics of concurrency, Information and Control, 54, pp 70-120, 1982.

[6] BROY, M., Fixed point theory for communication and concurrency, in Bjorner, Ed., IFIP TC 2 Working conference on Formal description of Programming Concepts II, Garmisch, June 1982, pp 125-147, North Holland, Amsterdam, 1983.

[7] BROY, M., BAUER, L., A systematic approach to language constructs for concurrent programs, in Science of Computer Programming 4, pp 103-139, North Holland, 1984.

[8] BURGE, W.H., Stream processing functions, in IBM journal of research and development, 19, pp 12-25, 1975.

[9] DARLINGTON, J., HENDERSON, P., TURNER, D.A. (eds), Functional programming and its applications, Cambridge University Press, Cambridge, 1982.

[10] DENNIS, J.B., WENG, K.K.-S., An abstract implementation for concurrent computation with streams, in Proc. 1979 int. conf. on parallel processing, pp 35-45, 1979.

[11] DUGUNDJI, J., Topology, Allyn and Bacon, Inc., Boston, 1966.

[12] IDA, T., TANAKA, J., Functional programming with streams, in Information Processing, pp 265-270, North-Holland, 1983.

[13] MEERTENS, L.G.L.T., Procedurele datastructuren (in Dutch), in Colloquium Capita Datastructuren (J.C. VAN VLIET (Red)), pp 171-186, MC syllabus 37, Amsterdam, 1978.

[14] MICHAEL, E., Topologies on spaces of subsets, in Trans. AMS 71, pp 152-182, 1951.

[15] NAKATA, I, Programming with streams, in IBM-research report , RJ 3751 (43317) , 1983.

[16] NIVAT, M., Infinite words, infinite trees, infinite computations, in Foundations of computer science III.2 (de Bakker, J.W., van Leeuwen, J. (eds)), 3-52, Mathematical centre tracts 109, 1979.

[17] ROUNDS, W.C., GOLSON, W.G., Connections between two theories of concurrency: Metric spaces and synchronization trees, Information and control 57, pp 102-124, 1983.

[18] SMYTH, M.B., Powerdomains and predicate transformers: a topological view, Proc. 10th ICALP, Barcelona, Spain, Diaz, J. (ed), LNCS 154, pp 662-676, 1983.

[19] TISON, S., DAUCHET, M., COMYN, G., Metrical and ordered properties of powerdomains, Proc. FCT conference, Borgholm, Sweden, Karpinski, M. (ed), LNCS 158, pp 465-474, 1983.

[20] TURNER, D.A., Recursion equations as a programming language, in Darlington, J., Henderson, P., Turner, D.A. (eds), Functional programming and its applications, Cambridge University Press, Cambridge, pp 1-28, 1982.

[21] TURNER, D.A, A new implementation technique for applicative languages, in Software- Practice and experience, 9 ,pp 31-49, 1979.

[22] WADGE, W.W., An extensional treatment of dataflow deadlock, in Theoretical Computer Science 13 (1981), pp 3-15, 1981.

[23] WADLER, P., Applicative style programming, program transformation and list operators, in Proc. functional programming languages and computer architecture, pp 25-32, 1981.