Abstract

The focus of this report is on program semantics. Over the years different approaches have been developed to model program semantics. This particular paper is restricted to denotational semantics. Two notions take a leading role in this paper, metric spaces and probabilistic language with non-termination. The probabilistic component is caused by Probabilistic Choice constructors, which choose to execute one program or another probabilistically. While the non-termination component is caused by recursive constructors, in particular the while. Metric spaces are an essential ingredient in the denotational model presented. The metric structure is exploited to model recursive constructors and to define operators over infinite structures. The main feature of denotational semantics is its compositionality. The semantics of a composite program is expressed in terms of the semantics of each of the programs that compose it. To deal with recursive constructors in the denotational environment, a fixed-point mathematical structure is necessary Traditionally, partially ordered sets have been used for this purpose. In contrast, in this report, we use metric spaces and Banach's Fixed Point Theorem as an alternative tool. Finally, it is demonstrated that the model presented in this report, based on metric spaces, is equivalent to an already known model of semantics based on partial order sets.

1 Preliminar definitions

In this section general theory over metric spaces is presented. This theory is applied to model denotational semantics of a probabilistic language.

Useful definitions to support an approach with metric spaces

We begin with the definition of the most basic notion, the metric space.

Definition 1.1 (Metric spaces)
A metric space is a pair $\langle X, d_X \rangle$ *tal que:*

- X *a non void space*
- d_X *a function* $d_X : X \times X \to [0,1]$ *such that*
  - *Identity of indiscernible:* $\forall x, y \in X, d_X(x,y) = 0 \iff x = y$
  - *Symmetry:* $\forall x, y \in X, d_X(x,y) = d_X(y,x)$
  - *Triangle inequality:* $\forall x, y, z \in X, d_X(x,z) \leq d_X(x,y) + d_X(y,z)$

Definition 1.2 (Binary Tree)
$T(A) ::= \mathsf{DNode}(T(A), v, T(A)) \mid \mathsf{Leaf}(v) \mid \mathsf{SNode}(T(A), v) : v \in A$

Being A any set, since it corresponds to a polymorphic binary tree

The definition of a binary tree structure is necessary to model the successive ramifications on the value of a probabilistic state generated by the application of *probabilistic choice*

Definition 1.3 (Dimension of the longest common prefix)
$lcp : T(A) \to T(A) \to \mathbb{N}$

$$lcp(t_a, t_b) = \begin{cases} 1 + min(lcp(t_1, t_3), lcp(t_2, t_4)) & t_a = \mathsf{DNode}(t_1, v, t_2) \wedge t_b = \mathsf{DNode}(t_3, v, t_4) \\ 1 & t_a = \mathsf{Leaf}(v) \wedge t_b = \mathsf{Leaf}(v) \\ 1 + lcp(t_1, t_2) & t_a = \mathsf{SNode}(t_1, v) \wedge t_b = \mathsf{SNode}(t_2, v) \\ 0 & otherwise \end{cases}$$

The longest common prefix between two binary trees corresponds to the maximum number of levels between two trees, such that all elements of each level are equal
This definition will make it possible to define a metric on the set of binary trees.

Definition 1.4 (Baire metric over Binary Trees)
$d_{T(A)} : T(A) \to T(A) \to [0,1]$

$$d_{T(A)}(t_1, t_2) = \begin{cases} 0 & t_1 = t_2 \\ 2^{-lcp(t_1,t_2)} & t_1 \neq t_2 \end{cases}$$

Such that $lcp(t_1, t_2)$ corresponds to the dimension of the longest common prefix between t_1 y t_2
This metric allows us to give the characteristic of metric space to the pair $\langle T(A), Baire\rangle$ which is demonstrated as follows

---

Proposition 1.1 ($\langle T(A), Baire\rangle$ is a metric space)

(1) PDQ $\forall t_1, t_2 \in T(\Theta), Baire(t_1, t_2) = 0 \iff t_1 = t_2$

We proceed to demonstrate by double implication

$(\Longleftarrow)$

$$\begin{array}{rl} & t_1 = t_2 \\ \Longrightarrow & \{Baire.1\} \\ & Baire(t_1, t_2) = 0 \end{array}$$

$(\Longrightarrow)$

$$\begin{array}{rl} & Baire(t_1, t_2) = 0 \\ \Longleftrightarrow & \{Baire.1\} \\ & 2^{-n} = 0 \\ \Longleftrightarrow & \exists n \in \mathbb{N} : 2^{-n} = 0 \\ \Longleftrightarrow & \to\leftarrow \end{array}$$

(2) PDQ $\forall t_1, t_2 \in T(\Theta), Baire(t_1, t_2) = Baire(t_2, t_1)$

We proceed to demonstrate by case analysis

$(t_1 = t_2)$

$$\begin{array}{rl} & Baire(t_1, t_2) \\ = & \{Baire.1\} \\ & 0 \\ = & \{Baire.1\},\ \{Hypothesis\ t_1 = t_2\} \\ & Baire(t_1, t_2) \end{array}$$

$(t_1 \neq t_2)$

$$\begin{array}{rl} & Baire(t_1, t_2) \\ = & \{Baire.2\} \\ & 2^{-lcp(t_1,t_2)} \\ = & \{lcp(t_1, t_2) = lcp(t_2, t_1)\} \\ & 2^{-lcp(t_2,t_1)} \\ = & \{Baire.2\} \\ & Baire(t_2, t_1) \end{array}$$

(3) $PDQ\ \forall t_1, t_2, t_3 \in T(\Theta), Baire(t_1, t_3) \leq Baire(t_1, t_2) + Baire(t_2, t_3)$

We proceed to demonstrate by case analysis
$(d(t_1, t_3) < d(t_1, t_2))$

It holds
(1) $d(t_2, t_3) = d(t_1, t_3)$
(2) $d(t_2, t_3) \geq d(t_1, t_2)$
(3) $d(t_1, t_3) \geq d(t_1, t_2)$

Then
$$d(t_1, t_3) \geq d(t_1, t_2)$$
$$\iff$$
$$2^{d(t_1,t_3)} \geq 2^{d(t_1,t_2)}$$
$$\iff$$
$$2^{-d(t_1,t_3)} \leq 2^{-d(t_1,t_2)}$$
$$\iff$$
$$2^{-d(t_1,t_3)} \leq 2^{-d(t_1,t_2)} + 2^{-d(t_2,t_3)}$$

$(d(t_1, t_3) > d(t_1, t_2))$

It holds
(1) $d(t_2, t_3) = d(t_1, t_2)$
(2) $d(t_2, t_3) \leq d(t_1, t_3)$
(3) $d(t_1, t_2) \leq d(t_1, t_3)$

Then
$$d(t_2, t_3) \leq d(t_1, t_3)$$
$$\iff$$
$$2^{d(t_2,t_3)} \leq 2^{d(t_1,t_3)}$$
$$\iff$$
$$2^{-d(t_2,t_3)} \geq 2^{-d(t_1,t_3)}$$
$$\iff$$
$$2^{-d(t_2,t_3)} + 2^{-d(t_1,t_2)} \geq 2^{-d(t_1,t_3)}$$
$$\iff$$
$$2^{-d(t_1,t_3)} \leq 2^{-d(t_1,t_2)} + 2^{-d(t_2,t_3)}$$

$(d(t_1, t_3) = d(t_1, t_2))$

$$
\begin{array}{ll}
 & \textit{It holds} \\
(1) & d(t_2, t_3) = d(t_1, t_3) \\
(2) & d(t_2, t_3) = d(t_1, t_2) \\
(3) & d(t_1, t_3) = d(t_1, t_2) \\
 & \\
 & \textit{Then} \\
 & d(t_1, t_3) = d(t_1, t_2) \\
\iff & \\
 & 2^{d(t_1,t_3)} = 2^{d(t_1,t_2)} \\
\iff & \\
 & 2^{-d(t_1,t_3)} = 2^{-d(t_1,t_2)} \\
\iff & \\
 & 2^{-d(t_1,t_3)} \leq 2^{-d(t_1,t_2)} + 2^{-d(t_2,t_3)}
\end{array}
$$

Definition 1.5 (Convergent succession)
A sequence $(x_n)_n$ in $\langle X, d_X \rangle$ is convergent if: $\forall \epsilon > 0 : \exists N \in \mathbb{N} : \forall n \geq N : d_X(x_n, x) \leq \epsilon$ *for some* $x \in X$

Definition 1.6 (Cauchy succession)
A sequence $(x_N)_n$ in $\langle X, d_X \rangle$ is a Cauchy sequence if: $\forall \epsilon > 0 : \exists N \in \mathbb{N} : \forall m, n \geq N : d_X(x_m, x_n) \leq \epsilon$

Definition 1.7 (Complete metric space)
A metric space $\langle X, d_X \rangle$ is defined complete if every Cauchy sequence $(x_n)_n$ in $\langle X, d_X \rangle$ is convergent

The definition of complete metric space is part of the Banach Theorem hypothesis, a theorem through which it can be stated that a transformer has a single fixed point, useful later, to define the semantics of the constructor while c do S

Proposition 1.2 (Proof that $\langle T(A), Baire \rangle$ is a complete metric space)

Let $(t_n)_n$ be a Cauchy sequence in $T(A)$, a branch of a binary tree. There are two possible cases

- *Let $(t_n)_n$ be an eventually constant sequence, that is, a terminating path from the unreduced program to its normal form, that is:*

$$\exists N \in \mathbb{N} : \forall n \geq N, t_n = t : t \in T(\Theta)$$

Then $(t_n)_n$ converges to t

- *Let $(t_n)_n$ be a sequence not eventually constant. By hypothesis, $(t_n)_n$ is Cauchy, then, given by the definition of Cauchy sequence, we can state, without loss of generality that:*

$$\forall n : Baire(t_n, t_{n+1}) \leq 2^{-n}$$

Given Baire's definition, one of the two following possibilities is true

1. $t_n = t_{n+1}$
2. $lcp(t_n, t_{n+1}) \geq n$

Since $(t_n)_n$ is eventually non-constant the depth of t_n is at least n. Let a_n be the n-th element of t_n. Then it can be verified that $(t_n)_n$ converges to $a_1 a_2 ...$

Definition 1.8 (Contractive function)
$f : X \to Y$ is contractive $\iff \forall x_1, x_2 \in X, d_Y(f(x_1), f(x_2)) \leq \alpha \cdot d_X(x_1, x_2) : \alpha \in [0, 1]$

Where d_X is a metric of X and d_y is a metric of Y.

This definition constitutes part of the hypothesis of Banach's Theorem, which will be presented below, and will allow to define, later on, the semantics of the constructor $\mathsf{while; c; do; S}$.

Theorem 1.1 (Banach Theorem)
Let $\langle X, d_X \rangle$ be a metric space, then:

- *$f : X \to X$ is contractive* $\implies ((f(x) = x \wedge f(y) = y) \implies x = y)$
- *($f : S \to X$ is contractive y $\langle X, d_X \rangle$ is a complete metric space)* $\implies \forall x_0 \in X, f(lim_n x_n = lim_n x_n)$*, such that $x_{n+1} = f(x_n)$*

Mathematical tools for defining a general probabilistic metric language

Definition 1.9 (Support of a function) *Let* $f : D \to [0,1]$
Then $support(f) = \{d \in D : f(d) > 0\}$

Definition 1.10 (Set of probabilistic pseudo-distributions)
$Dist(X) = \{f \in (X \to [0,1]) : \Sigma_{x \in X} f(x) \leq 1 \wedge$ *support(f) is countable*$\}$

Given an element $f \in Dist(X)$ and an element $x \in X$, $f(x)$ is interpreted as the probability of ocurrence of x.

2 Language definition

This section introduces the L_{pw} language and its denotational semantics, both metric and cpo based The L_{pw} language is a basic programming language that contains the constructors skip (skip), assignment ($\mathsf{x} := \mathsf{e}$), sequential composition ($\mathsf{S_1}; \mathsf{S_2}$), conditional if ($\mathsf{if\ c\ then\ S_1\ else\ S_2}$), *probabilistic choice* ($\mathsf{S_1} \oplus_{\mathsf{p}} \mathsf{S_2}$) and a loop *while* ($\mathsf{while\ c\ do\ S}$).

The L_{pw} language programs are interpreted as state transformers, where the state corresponds to the assignment of natural values to variables..

- The skip operator does not generate any change of state.
- The value of a variable in a state can be modified by means of the assignment operator :=. An assignment corresponds to a program $\mathsf{x} := \mathsf{e}$, which modifies the value associated to the variable x, by the value of the arithmetic expression e
- The sequential composition operator ($\mathsf{S_1}; \mathsf{S_2}$), sequentially executes two programs starting from an initial state, first S_1 and then S_2
- The conditional *if* uses Boolean expressions to choose between the application of one or the other program in an exclusive manner
- The operator *probabilistic choice* ($\mathsf{S_1} \oplus_{\mathsf{p}} \mathsf{S_2}$) executes program S_1 with probability p and program S_2 with complementary probability $(1 - p)$
- The operator *while* ($\mathsf{while\ c\ do\ S}$), executes the program S as long as the condition c is fulfilled.

The mathematical formalization of these ideas is developed as follows.

Definition 2.1 (Program variables) Var *corresponds to the set of variable symbols used by programs.*
The elements of the set Var *will be denoted by* x*.*

Definition 2.2 (Deterministic state) $\Sigma = (\mathsf{Var} \to \mathbb{N})$ *corresponds to the set of mapping functions, which associate a natural value to each program variable. The elements of the set* Σ *will be denoted by* σ*.*

Definition 2.3 (Probabilistic state) *For a deterministic state, each variable in* Var *is mapped to a single value in* $\mathbb{N}$*. However, for a probabilistic state, each variable is mapped to a distribution in* $\mathbb{N}$*. The set of probabilistic states is then defined as* $\Theta = Dist(\Sigma)$*. The elements of the set* Θ *will be denoted by* θ*.*

Definition 2.4 (Arithmetical expressions) $\mathsf{Expr}(\mathsf{Var})$ *corresponds to the syntactic category of arithmetic expressions. The dependency on Var given as a type argument indicates that these arithmetic expressions use program variables in their construction.*

This set is described as:
$e ::= v \mid n \mid e + e \mid e - e \mid e \cdot e \mid e \; div \; e \mid e \; mod \; e \; ...$

The evaluation function of $\mathsf{Expr}(\mathsf{Var})$ *corresponds to the semantic function:* $\mathcal{E} : \mathsf{Expr}(\mathsf{Var}) \to \Sigma \to \mathbb{N}$ *which computes the abstract natural value of an expression given the associated values of each variable according to a state.*

Definition 2.5 (Boolean Expressions) $\mathsf{BExpr}(\mathsf{Var})$ *corresponds to the syntactic category of Boolean expressions. The dependence on Var, given as a type argument, indicates that these Boolean expressions use program variables in their construction.*

This set is described as:
$c ::= true \mid false \mid e = e \mid e < e \; ... \mid c \wedge c \mid c \wedge c \mid \neg c$

The evaluation function of $\mathsf{BExpr}(\mathsf{Var})$ *corresponds to the semantic function:* $\mathcal{B} : \mathsf{BExpr}(\mathsf{Var}) \to \Sigma \to \{tt, ff\}$ *which computes the abstract boolean value of and expression given the associated values of each variable according to a state*

For the purposes of this report, tt and ff correspond to the abstract values true and false respectively. While true and false correspond to concrete values of the L_{pw} *language.*

Definition 2.6 (Programs) Stat *corresponds to the syntactic category of programs, whose meaning corresponds to a state transformer. This syntactic category is given by:*

$$S ::= \mathsf{skip} \mid \mathsf{x} := \mathsf{e} \mid \mathsf{S};\mathsf{S} \mid \mathsf{S} \oplus_{\mathsf{p}} \mathsf{S} \mid \mathsf{if\ c\ then\ S\ else\ S} \mid \mathsf{while\ c\ do\ S}$$

The program $\mathsf{if;c\ then\ S_1\ else\ S_2}$ *evaluates the Boolean expression c by using the semantic function* $\mathcal{B}$*, if its value is tt, then it executes the program* S_1*, on the other hand, if its value is ff, it executes the program* S_2*. The program* $\mathsf{while\ c\ do\ S}$ *executes program S as long as the value of the Boolean expression c is tt. If the value of the Boolean expression c is ff, then it performs a skip.*

3 Denotational semantics of the language L_{pw}

The denotational semantics assigns to each program a probabilistic state mapping function and a probabilistic state tree. This tree structure models the nondeterminism generated by the constructor *probabilistic choice*, by fragmenting the two possible execution paths in each of its occurrences
The meaning of a program of L_{pw} is then modeled by a denotational semantic function $D : \mathsf{Stat} \to \Theta \to T(\Theta)$
The formalization of this model is then carried out.

Semantics of terminating probabilistic programs

The terminating probabilistic programs correspond to those using the constructors (skip), ($\mathsf{x} := \mathsf{e}$), ($\mathsf{S};\mathsf{S}$), ($\mathsf{S} \oplus_{\mathsf{p}} \mathsf{S}$) y ($\mathsf{if\ c\ then\ S\ else\ S}$)

The treatment of the constructor $\mathsf{while;c;do;S}$ will be discussed in a separate subsection, since the definition of its semantics, unlike the previous constructors, uses Banach's Fixed Point Theorem and deserves a more extensive and particular work.

Definition 3.1 (Variation of a deterministic state)
Given $x \in Var, v \in N, s \in \Sigma$

$$s[x/v](x') = \begin{cases} v & x = x' \\ s(x') & otherwise \end{cases}$$

Corresponds to the deterministic state updated by modifying the mapping of x to a new value v in $\mathbb{N}$

This function allows you to define the update of a probabilistic state when the mapping of a variable is modified

---

Definition 3.2 (Variation of a probabilistic state)
Given $x \in Var, f : (\mathsf{Stat} \to \mathbb{N}), \theta \in \Theta, \sigma \in \Sigma$

$$\theta[x/f](\sigma) = \Sigma_{\sigma' : \sigma'[x/f(\sigma')]=\sigma} \theta(\sigma')$$

The variation of θ *corresponds to the updated probabilistic state when modifying the mapping of* x *to a new distribution over* $\mathbb{N}$

This function allows the definition of the denotational semantics of the constructor $:=$

Definition 3.3 (Concatenation of trees)
$(;) : T(\Theta) \to (\Theta \to T(\Theta)) \to T(\Theta)$

$$t; f = \begin{cases} \mathsf{DNode}((t_1; f), \theta, (t_2; f)) & t = \mathsf{DNode}(t_1, \theta, t_2) \\ \mathsf{SNode}(f(\theta), \theta) & t = \mathsf{Leaf}(\theta) \\ \mathsf{SNode}((t_1; f), \theta) & t = \mathsf{SNode}(t_1, \theta) \end{cases}$$

This function allows the definition of the denotational semantics of the constructor ;

Given a probabilistic state tree $t : T(\Theta)$ *and a probabilistic state function to probabilistic state tree* $f : (\Theta \to T(\Theta)$*, extends each leaf by adding the result of applying* f *to the value contained in each leaf*

Definition 3.4 (Unscaled conditional)
$c? : \Theta \to \Theta$

$$c?\theta(s) = \begin{cases} \theta(s) & \mathcal{B}(c)(s) = true \\ 0 & otherwise \end{cases}$$

This function allows, given the value of a Boolean expression, to filter out any value of a distribution, such that it meets or does not meet the condition. Useful for the definition of denotational semantics of the constructor if

---

Semantics of probabilistic programs with non-termination

These programs correspond to those with the use of at least one while constructor.
The denotational semantics of the recursive constructor while is obtained according to its single fixed point
According to the above, it is convenient to formalize the set of probability distribution trees as a metric space satisfying Banach's Theorem

---

Definition 3.5 (Tree of probabilistic states)

$T(\Theta) ::= \mathsf{DNode}(T(\Theta), v, T(\Theta)) \mid \mathsf{Leaf}(v) \mid \mathsf{SNode}(T(\Theta), v)\ tq\ v \in \Theta$

This corresponds to a particular case of a polymorphic binary tree, such that it is defined over the set of state distributions Θ

Definition 3.6 (Dimension of the longest common prefix)
This definition makes sense in the following definition, as it is part of Baire's definition of metrics
$lcp : T(A) \to T(A) \to \mathbb{N}$

$$lcp(t_a, t_b) = \begin{cases} 1 + min(lcp(t_1, t_3), lcp(t_2, t_4)) & t_a = \mathsf{DNode}(t_1, v, t_2) \wedge t_b = \mathsf{DNode}(t_3, v, t_4) \\ 1 & t_a = \mathsf{Leaf}(v) \wedge t_b = \mathsf{Leaf}(v) \\ 1 + lcp(t_1, t_2) & t_a = \mathsf{SNode}(t_1, v) \wedge t_b = \mathsf{SNode}(t_2, v) \\ 0 & otherwise \end{cases}$$

Definition 3.7 (Baire metric over probabilistic state trees)
$d_{T(\Theta)} : T(\Theta) \to T(\Theta) \to [0, 1]$

$$d_{T(\Theta)}(t_1, t_2) = \begin{cases} 0 & t_1 = t_2 \\ 2^{-lcp(t_1, t_2)} & t_1 \neq t_2 \end{cases}$$

Such that $lcp(t_1, t_2)$ corresponds to the longest common prefix dimension between t_1 and t_2.

Definition 3.8 (Unfold constructor while**)**

$\mathcal{D}($while c S$)$

$= \{$*First order unfold*$\}$

$\mathcal{D}($if c then (S; while c do S) else skip$)$

$=$ $\{$*Semantic* $\mathcal{D}$(if c then S_1 else S_2)$\}$

$\mathsf{DNode}(\mathcal{D}($S; while c do S$)(c?\theta), \theta, \mathcal{D}(\mathsf{skip})(\neg c?\theta))$

$=$ $\{$*Semantic* $\mathcal{D}(\mathsf{S}_1; \mathsf{S}_2)\}$

$\mathsf{DNode}(\mathcal{D}(\mathsf{S})(c?\theta); \mathcal{D}($while c do S$), \theta, \mathcal{D}(\mathsf{skip})(\neg c?\theta))$

$=$ $\{$*Semantic* $\mathcal{D}(\mathsf{skip})\}$

$\mathsf{DNode}(\mathcal{D}(S)(c?\theta); \mathcal{D}(\mathsf{while\ c\ do\ S}), \theta, \mathsf{Leaf}(\neg c?\theta))$

Definition 3.9 (Transformer of constructor while**)**
We define a transformer Φ
$\Phi : (\Theta \to T(\Theta)) \to (\Theta \to T(\Theta))$

$$\Phi(\phi)(\theta) = \mathsf{DNode}(\mathcal{D}(S)(c?\theta); \phi, \theta, \mathsf{Leaf}(\neg c?\theta))$$

Lemma 3.1 (Transformer of constructor while **is contractive)**

$$\forall \phi_1, \phi_2 \in (\Theta \to T(\Theta)), d_{T(\Theta)}(\Phi(\phi_1), \Phi(\phi_2)) \leq d_{T(\Theta)}(\phi_1, \phi_2)$$

Proposition 3.1 (Proof of the lemma)
It holds:

$d_{T(\Theta)}(\Phi(\phi_1), \Phi(\phi_2))$

$=$ $\{def\ \Phi\}$

$d_{T(\Theta)}(\mathsf{DNode}(\mathcal{D}(S)(c?\theta); \phi_1, \theta, \mathsf{Leaf}(\neg c?\theta)), \mathsf{DNode}(\mathcal{D}(S)(c?\theta); \phi_2, \theta, \mathsf{Leaf}(\neg c?\theta)))$

$=$ $\{Baire.2\}$

$2^{-lcp(\mathsf{DNode}(\mathcal{D}(S)(c?\theta);\phi_1,\theta,\mathsf{Leaf}(\neg c?\theta),\mathsf{DNode}(\mathcal{D}(S)(c?\theta);\phi_2,\theta,\mathsf{Leaf}(\neg c?\theta))}$

$=$ $\{lcp.1\}$

$2^{-(1+min(lcp(\mathcal{D}(S)(c?\theta);\phi_1,\mathcal{D}(S)(c?\theta);\phi_2),lcp(\mathsf{Leaf}(\neg c?\theta),\mathsf{Leaf}(\neg c?\theta))))}$

$=$ $\{lcp.2\}$

$2^{-(1+min(lcp(\mathcal{D}(S)c?\theta);\phi_1,\mathcal{D}(S)(c?\theta);\phi_2),1)))}$

$=$ $\{$*Property of powers:* $2^{a+b} = 2^a \cdot 2^b\}$

$2^{-1} \cdot 2^{-min(lcp(\mathcal{D}(S)(c?\theta);\phi_1,\mathcal{D}(S)(c?\theta);\phi_2),1))}$

$=$ $\{$*Relation between functions min and máx subject to* $lcp \geq 0\}$

$2^{-1} \cdot max(2^{-lcp(\mathcal{D}(S)(c?\theta);\phi_1,\mathcal{D}(S)(c?\theta);\phi_2)}, 2^{-1})$

$=$ $\{$*Def Baire.2*$\}$

$2^{-1} \cdot max(d(\mathcal{D}(S)(s?\theta);\phi_1, \mathcal{D}(S)(c?\theta);\phi_2), \frac{1}{2})$

$=$ $\{$*Proposition 3.1*$\}$

$\frac{1}{2} \cdot max(r, \frac{1}{2})$ *such that* $r \leq \frac{1}{2}d(\phi_1, \phi_2)$

$\leq$ $\{d(\phi_1, \phi_2) \geq 0$ *then greater than* $\frac{1}{2}\}$

$\frac{1}{2} \cdot \frac{1}{2}d(\phi_1, \phi_2)$

$\leq$ $\{$*In particular*$\}$

$\frac{1}{2}d(\phi_1, \phi_2)$

Definition 3.10 (Semantic of constructor while**)**
Fulfilling then the hypotheses of Banach's theorem, it can be stated that the transformer

$$\Phi(\phi)(\theta) = \mathsf{DNode}(\mathcal{D}(S)(c?\theta), \theta, \mathsf{Leaf}(\neg c?\theta))$$

It has a single fixed point and consequently the semantics of the constructor while *is* $\mathcal{D}(\mathsf{while\ c\ do\ S}) = fix(\Phi)$

Equivalence between the two models of denotational semantics

Definition 3.11 (Language semantics using metric spaces)
The semantic function is defined as $D : \mathsf{Stat} \to \Theta \to T(\Theta)$

$$
\begin{array}{lcl}
\mathcal{D}(\mathsf{skip})(\theta) & = & \mathsf{Leaf}(\theta) \\
\mathcal{D}(\mathsf{x} := \mathsf{e})(\theta) & = & \mathsf{Leaf}(\theta[x, \mathcal{E}(e)] \\
\mathcal{D}(\mathsf{S_1}; \mathsf{S_2})(\theta) & = & \mathcal{D}(\mathsf{S_1})(\theta); \mathcal{D}(\mathsf{S_2}) \\
\mathcal{D}(\mathsf{S_1} \oplus_{\mathsf{p}} \mathsf{S_2})(\theta) & = & \mathsf{DNode}(\mathcal{D}(\mathsf{S_1})(p \cdot \theta), \theta, \mathcal{D}(\mathsf{S_2})((1-p) \cdot \theta)) \\
\mathcal{D}(\mathsf{if\ c\ then\ S_1\ else\ S_2})(\theta) & = & \mathsf{DNode}(\mathcal{D}(\mathsf{S_1})(c?\theta), \theta, \mathcal{D}(\mathsf{S_2})(\neg c?\theta)) \\
\mathcal{D}(\mathsf{while\ c\ do\ S})(\theta) & = & fix(\Phi(\phi))(\theta)\ tq\ \Phi(\phi) = \mathsf{DNode}(\mathcal{D}(S)(c?\theta); \phi, \theta, \mathsf{Leaf}(\neg c?\theta))
\end{array}
$$

Definition 3.12 (Language semantics using cpo)

The semantic function is defined as $D : \mathsf{Stat} \to \Theta \to \Theta$

$$
\begin{array}{lcl}
\mathcal{D}(\mathsf{skip})(\theta) & = & \theta \\
\mathcal{D}(\mathsf{x} := \mathsf{e})(\theta) & = & \theta[x/\mathcal{E}(e)] \\
\mathcal{D}(\mathsf{S_1}; \mathsf{S_2})(\theta) & = & \mathcal{D}(S_2)(\mathcal{D}(\mathsf{S_1})(\theta)) \\
\mathcal{D}(\mathsf{S_1} \oplus_{\mathsf{p}} \mathsf{S_2})(\theta) & = & \mathcal{D}(\mathsf{S_1})(\theta) \oplus_p \mathcal{D}(\mathsf{S_2})(\theta) \\
\mathcal{D}(\mathsf{if\ c\ then\ S_1\ else\ S_2})(\theta) & = & \mathcal{D}(\mathsf{S_1})(c?\theta) + \mathcal{D}(\mathsf{S_2})(\neq c?\theta) \\
\mathcal{D}(\mathsf{while\ c\ do\ S})(\theta) & = & fix(\Phi_{\langle c,S\rangle}(\phi))\ tq\ \Phi_{\langle c,S\rangle}(\phi)(\theta) = \phi(\mathcal{D}(S)(c?\theta)) + \neq c?\theta
\end{array}
$$

The distribution given by the denotational semantics determined by cpo corresponds to the aggregation by summation of the distributions on the leaves of the denotational semantics given by metric spaces. Then the following operator is introduced

---

Definition 3.13 (Adding tree leaves)
$aggregate : T(\Theta) \to \Theta$

$$
aggregate(t) = \begin{cases} aggregate(t_1) + aggregate(t_2) & t = DNode(t_1, \theta, t_2) \\ aggregate(t_1) & t = SNode(t_1, \theta) \\ \theta & t = Leaf(\theta) \end{cases}
$$

Proposition 3.2 (Equivalence between cpo model and metric model)
We proceed to prove by structural induction
Let D_m *be the denotational semantic function defined by metric spaces and let* D_c *be the denotational semantic function defined by cpo*

Through demonstration, the inductive hypothesis is used:

$$\forall \theta \in \Theta, aggregate(\mathcal{D}_m(S_1)(\theta)) = \mathcal{D}_c(S_1)(\theta) \wedge aggregate(\mathcal{D}_m(S_2)(\theta)) = \mathcal{D}_c(S_2)(\theta)$$

In addition, the following property is used, which is called "Note":

$$\mathsf{aggregate}(\mathcal{D}_\mathsf{m}(\mathsf{S})(\mathsf{aggregate}(\mathsf{t}))) = \mathsf{aggregate}(\mathsf{t}; \mathcal{D}_\mathsf{m}(\mathsf{S})$$

The following is a proof of equivalence between the two semantics for each language constructor.
To identify whether the semantic function $\mathcal{D}$ corresponds to the one defined by metric spaces or by cpo, it will be denoted as follows $\mathcal{D}_m$ and $\mathcal{D}_c$ respectively

$(Case\ S = \mathsf{skip})$

$$\begin{array}{rl} & aggregate(\mathcal{D}_m(\mathsf{skip})(\theta)) \\ = & \{Semántica\ \mathcal{D}_m(\mathsf{skip})\} \\ & aggregate(Leaf(\theta)) \\ = & \{aggregate\} \\ & \theta \\ = & \{Semantic\ \mathcal{D}_c(\mathsf{skip})\} \\ & D_c(\mathsf{skip})(\theta) \end{array}$$

$(Case\ S = \mathsf{x} := \mathsf{e})$

$$\begin{array}{rl} & aggregate(\mathcal{D}_m(x := e)(\theta)) \\ = & \{Semantic\ \mathcal{D}_m(:=)\} \\ & aggregate(\mathsf{Leaf}(\theta)[x, \mathcal{E}(e)]) \\ = & \{aggregate\} \\ & \theta[x, \mathcal{E}(e)] \\ = & \{Semantic\ \mathcal{D}_c(:=)\} \\ & \mathcal{D}_c(x := e)(\theta) \end{array}$$

$(Case\ S = \mathsf{S}_1; \mathsf{S}_2)$

$$
\begin{array}{ll}
 & It\ holds \\
 & aggregate(\mathcal{D}_m(\mathsf{S}_1; \mathsf{S}_2)) \\
= & \{Semantic\ \mathcal{D}_m(\mathsf{S}_1; \mathsf{S}_2)\} \\
(1) & aggregate(\mathcal{D}_m(\mathsf{S}_1)(\theta); \mathcal{D}(\mathsf{S}_2)
\end{array}
$$

$$
\begin{array}{ll}
 & Also,\ it\ holds \\
 & \mathcal{D}_c(\mathsf{S}_1; \mathsf{S}_2)(\theta) \\
= & \{Semantic\ \mathcal{D}_c(\mathsf{S}_1; \mathsf{S}_2)\} \\
 & \mathcal{D}_c(\mathsf{S}_2)(\mathcal{D}_\mathsf{c}(\mathsf{S}_1)(\theta)) \\
= & \{Inductive\ hypothesis\} \\
 & \mathcal{D}_c(\mathsf{S}_2)(\mathsf{aggregate}(\mathcal{D}_\mathsf{m}(\mathsf{S}_1)(\theta))) \\
= & \{Inductive\ hypothesis\} \\
(2) & aggregate(\mathcal{D}_m(\mathsf{S}_2)(\mathsf{aggregate}(\mathcal{D}_\mathsf{m}(\mathsf{S}_1)(\theta))))
\end{array}
$$

We look for both expressions, (1) and (2) to be equal. For the above, we evaluate three subcases, which correspond to the cases of the function ;

$(Subcase\ \mathcal{D}_m(\mathsf{S}_1)(\theta) = \mathsf{Leaf}(\theta))$

$$
\begin{array}{ll}
 & aggregate(\mathcal{D}_m(\mathsf{S}_2)(aggregate(\mathsf{Leaf}(\theta)))) \\
= & \{aggregate.3\} \\
 & aggregate(\mathcal{D}_m(\mathsf{S}_2)(\theta)) \\
= & \{aggregate.2\} \\
 & aggregate(SNode(\mathcal{D}_m(\mathsf{S}_2)(\theta), \theta)) \\
= & \{;.2\} \\
 & aggregate(\mathsf{Leaf}(\theta); \mathcal{D}_m(\mathsf{S}_2))
\end{array}
$$

$(Subcase\ \mathcal{D}_m(\mathsf{S}_1)(\theta) = \mathsf{SNode}(\mathsf{t}_1, \theta))$

$$
\begin{array}{ll}
 & aggregate(\mathcal{D}_m(\mathsf{S}_2)(aggregate(\mathsf{SNode}(\mathsf{t}_1, \theta)))) \\
= & \{aggregate.2\} \\
 & aggregate(\mathcal{D}_m(\mathsf{S}_2)(aggregate(t_1))) \\
= & \{Note\} \\
 & aggregate(t_1; \mathcal{D}_m(\mathsf{S}_2)) \\
= & \{aggregate.2\} \\
 & aggregate\mathsf{SNode}((\mathsf{t}_1; \mathcal{D}_\mathsf{m}(\mathsf{S}_2)), \theta)) \\
= & \{;.3\} \\
 & aggregate(\mathsf{SNode}(\mathsf{t}_1, \theta); \mathcal{D}_\mathsf{m}(\mathsf{S}_2))
\end{array}
$$

($Subcase\ \mathcal{D}_m(\mathsf{S_1})(\theta) = \mathsf{DNode}(\mathsf{t_1}, \theta, \mathsf{t_2})$)

$$
\begin{array}{ll}
 & aggregate(\mathcal{D}_m(\mathsf{S_2})(aggregate(\mathsf{DNode}(\mathsf{t_1}, \theta, \mathsf{t_2}))))\\
= & \{aggregate.1\}\\
 & aggregate(\mathcal{D}_m(\mathsf{S_2})(aggregate(t_1) + aggregate(t_2)))\\
= & \{Distributivity\}\\
 & aggregate((\mathcal{D}_m(\mathsf{S_2})(aggregate(t_1))) + (\mathcal{D}_m(\mathsf{S_2})(aggregate(t_2)))\\
= & \{Note\}\\
 & aggregate(t_1; \mathcal{D}_m(\mathsf{S_2})) + aggregate(t_2; \mathcal{D}_m(\mathsf{S_2}))\\
= & \{aggregate.1\}\\
 & aggregate(\mathsf{DNode}((\mathsf{t_1}; \mathcal{D}_\mathsf{m}(\mathsf{S_2})), \theta, (\mathsf{t_2}; \mathcal{D}_\mathsf{m}(\mathsf{S_2}))))\\
= & \{;.1\}\\
 & aggregate(\mathsf{DNode}(\mathsf{t_1}, \theta, \mathsf{t_2}); \mathcal{D}_\mathsf{m}(\mathsf{S_2})
\end{array}
$$

($Case\ S = \mathsf{if\ c\ then\ S_1\ else\ S_2}$)

$$
\begin{array}{ll}
 & aggregate(\mathcal{D}_m(\mathsf{if\ c\ then\ S_1\ else\ S_2})(\theta))\\
= & \{Semantic\ \mathcal{D}_m(\mathsf{if\ c\ then\ S_1\ else\ S_2})\}\\
 & aggregate(DNode(\mathcal{D}_m(S_1)(c?\theta), \theta, \mathcal{D}_m(S_2)(\neq c?\theta)))\\
= & \{aggregate\}\\
 & aggregate(\mathcal{D}_m(S_1)(c?\theta) + aggregate(\mathcal{D}_m(S_2)(\neg c?\theta)))\\
= & \{Inductive\ hypothesis\}\\
 & \mathcal{D}_c(S_1)(c?\theta) + aggregate(\mathcal{D}_m(S_2(\neg c?\theta))\\
= & \{Inductive\ hypothesis\}\\
 & \mathcal{D}_c(S_1)(c?\theta) + \mathcal{D}_c(S_2)(\neg c?\theta)\\
= & \{Semantic\ \mathcal{D}_m(\mathsf{if\ c\ then\ S_1\ else\ S_2})\}\\
 & \mathcal{D}_c(\mathsf{if\ c\ then\ S_1\ else\ S_2})(\theta)
\end{array}
$$

($Case\ S = \mathsf{S_1} \oplus_\mathsf{p} \mathsf{S_2}$)

$$
\begin{array}{ll}
 & aggregate(\mathcal{D}_m(S_1 \oplus_p S_2)(\theta))\\
= & \{Semantic\ \mathcal{D}_m(S_1 \oplus_p S_2)\}\\
 & aggregate(DNode(\mathcal{D}_m(S_1(p\theta), \theta, \mathcal{D}_m(S_2)((1-p)\theta)))\\
= & \{aggregate\}\\
 & aggregate(\mathcal{D}_m(S_1(p\theta)) + aggregate(\mathcal{D}_m(S_2((1-p)\theta))\\
= & \{Inductive\ hypothesis\}\\
 & \mathcal{D}_c(S_1)(p\theta)
\end{array}
$$

$(Case\ S = \mathsf{while\ c\ do\ S})$

$$
\begin{array}{rl}
 & \mathcal{D}_c(\mathsf{while\ c\ do\ S}) \\
= & \{Semantic\ \mathcal{D}_c(\mathsf{while\ c\ do\ S})\} \\
 & fix(\Phi_c(\phi)) \\
= & \{Definition \Phi_c\} \\
 & \theta.\phi(\mathcal{D}_c(S)(c?\theta)) + \neg c?\theta \\
= & \{aggregate.3\} \\
 & \theta.\phi(\mathcal{D}_c(S)(c?\theta)) + aggregate(\mathsf{Leaf}(\neg c?\theta)) \\
= & \{Semantic\ \mathcal{D}_c(\mathsf{S_1; S_2})\}\{Sea\mathsf{S}_\phi : \mathcal{D}(\mathsf{S}_\phi) = \phi\} \\
 & \theta.\mathcal{D}_c(\mathsf{S; S}_\phi)(c?\theta) + aggregate(\mathsf{Leaf}(\neg c?\theta)) \\
= & \{inductive\ hypothesis\} \\
 & \theta.aggregate(\mathcal{D}_m(\mathsf{S; S}_\phi)(c?\theta)) + aggregate(\mathsf{Leaf}(\neg c?\theta)) \\
= & \{Semantic\ \mathcal{D}(S_1; S_2)\} \\
 & \theta.aggregate(\mathcal{D}_m(\mathsf{S})(c?\theta); \mathcal{D}_m(\mathsf{S}_\phi)) + aggregate(\mathsf{Leaf}(\neq c?\theta)) \\
= & \{\mathcal{D}_m(S_\phi) = \phi\} \\
 & \theta.aggregate(\mathcal{D}_m(\mathsf{S})(c?\theta); \phi) + aggregate(\mathsf{Leaf}(\neg c?\theta)) \\
= & \{aggregate/1\} \\
 & \theta.aggregate(\mathsf{DNode}(\mathcal{D}_m(c?\theta); \phi), \theta, \mathsf{Leaf}(\neg c?\theta))) \\
= & \{Definition\ \Phi_m\} \\
 & aggregate(fix(\Phi_m(\phi))) \\
= & \{Semantic\ \mathcal{D}_c(\mathsf{while\ c\ do\ S})\} \\
 & aggregate(\mathcal{D}_m(\mathsf{while\ c\ do\ S}))
\end{array}
$$