Rijksuniversiteit Groningen Faculteit der Wiskunde en Natuurwetenschappen

Vakgroep Informatica

Bibl

Contraction based proof methods for a delay-insensitive algebra

Willem C. Mallon

Supervisor:
Dr. Ir. Jan Tijmen Udding

29th August 1995

RuG

Abstract

We define a metric for delay insensitive processes. We derive metric properties of operators in a delay insensitive algebra, and use these properties to formalize the notion of "input guardedness". We also derive a method to verify whether a function composed of operators is contracting with regard to its process arguments.

We define two relations on processes, and use these relations to derive proof methods for proving equality and refinement. Both proof methods can be used in linear derivations.

We present the proof methods in a simple format, that can easily be implemented in proof verification tools.

We give an example of the proof method for proving equality by the derivation of a component based design of an n-place delay-insensitive buffer.

Contents

1 Introduction and motivation 4

2 Processes 4
- 2.1 Introduction 4
- 2.2 Channels and communication. A formal model 4
- 2.3 Failure sets 5
  - 2.3.1 Introduction 5
  - 2.3.2 Operational view of failure sets 5

3 Delay insensitivity and receptiveness 6
- 3.1 introduction 6
- 3.2 Failure sets 6

4 A metric for failure sets 8
- 4.1 Introduction 8
- 4.2 ρ is an ultra metric 9
- 4.3 Metrics in vector space 9

5 The DI-algebra 9
- 5.1 introduction 9
- 5.2 Guardedness and contraction 10
- 5.3 fixpoints and recursive equations 11
- 5.4 uniqueness of fixpoints 12

6 A contraction based proof method 12
- 6.1 introduction 12
- 6.2 A new equivalence relation 13
  - 6.2.1 definition 13
  - 6.2.2 properties 13
- 6.3 Contraction based proof methods 14
  - 6.3.1 A method to prove equality 14
  - 6.3.2 A method to prove refinement 14

7 Example 14
- 7.1 introduction 14
- 7.2 syntax and rewrite rules 15
- 7.3 definitions and intentions 15
- 7.4 proof 16

8 Conclusions and further research 20

9 Acknowledgements 20

A Definition of operators 22

B The *disc* proofs 22
- B.1 introduction 22
- B.2 proof 22

C Derivation of a proof method for refinement 27
- C.1 Preliminaries 27
- C.2 The new relation 27
- C.3 relation to $\sqsubseteq$ 27
- C.4 Effect of application of input guarded functions 28

C.5 Properties of $\sqsubseteq_{xy}$ 28
C.6 New proof method 29
C.7 abstract example 30

D Some di-algebra rewrite rules **30**

1 Introduction and motivation

Asynchronous (as opposed to synchronous) circuits are currently under world-wide investigation, since popular opinion states that they might be energy saving, and even speed efficient in some cases. These alleged properties have been discussed extensively, and we refer to [8] for references. We usually model circuits by means of processes.

For the specification and comparison of asynchronous processes, a number of algebras have been developed, and one of them is the delay-insensitive algebra, that describes the most robust class of asynchronous processes.

During the development of a proof assistant for the delay insensitive algebra [2], it became apparent that there was no formal definition nor fundament for the rewrite rule that was referred to as "induction". After some discussion by the members of the study group for formal aspects of parallelism and program design, it seemed that a metric on processes might provide the fundament for this rewrite rule.

In this thesis, we will define a metric on processes, and use that metric to derive proof methods for the delay insensitive algebra. We will also show how these proof methods can be used in practice by a large derivation in the delay insensitive algebra.

In chapter 2 we introduce processes, and how we regard them as sets of communication sequences. In chapter 3 we introduce receptiveness and delay-insensitivity, and give the formal definition of our processes. In chapter 4 we introduce a metric on the space of processes. In chapter 5 we introduce the delay insensitive algebra, and derive metric properties of some of its operators. In chapter 6 we develop the proof methods. In chapter 7 we will give an example of the use of a proof method in practice.

2 Processes

2.1 Introduction

To us, a process is an object that can interact with its environment. By this we mean that the environment affects the process, and the process affects the environment. An example might be a glass of wine. The environment can push it, thus affecting the glass, and the glass can tip and colour a blouse, thus affecting the environment. Almost any object studied in physics might be called a process.

A first abstraction we make, is that we refrain from details of the inner workings of a process. We are only interested in a process's reaction to its environment. This is an abstraction that we typically make in every day's life. Whether an instant camera works by means of a complicated chemical reaction, or by means of a fast painting imp[9] is of no concern to us. We just want to obtain a picture after pressing a button. This abstraction is sometimes called the "black box principle".

Since we want to describe processes formally, we need to describe communication formally. We will assume that processes communicate by sending and receiving signals through channels.

The shape that signals and channels take, is not of much importance. It is just a means for describing communications. If we observe a coffee-machine for example, we could regard the slot as a channel, and a coin as a signal. However, the most convenient way to look at channels and signals is as wires and voltage changes, as our model is mainly meant to be a model for chip-design, where communications work that way.

A process can send signals through channels to the environment, and the environment can send signals through channels to the process. We assume channels to be unidirectional, so a channel can only be used by the process to send signals to the environment, or only be used by the environment to send signals to the process.

2.2 Channels and communication. A formal model

We mark every channel through which communication with a process is possible with a label, and name the set of these labels the alphabet. We usually denote the alphabet with A. We demand the set A to be finite.

The channels through which a process can receive signals, are collected in the set I (inputs). The channels through which a process can send signals, are collected in the set O (outputs). Since a channel is unidirectional it can not be in both sets at once.

We are only interested in processes with which we can actually communicate. Therefore, we demand:

$$A = I \cup O \ \wedge \ I \cap O = \emptyset \ \wedge \ I \neq \emptyset \ \wedge \ O \neq \emptyset$$

If the environment sends a signal through channel a to the process, we denote this event with $a?$. If the environment receives a signal through cannel b we denote this event with $b!$. Note that we observe interaction from the environment's point of view.

2.3 Failure sets

2.3.1 Introduction

If an environment is communicating with the process, we can write down which signals have been received and sent. We call such a sequence a *trace*. A trace is a finite sequence of communications. The inputs in a trace are sent by the environment, the outputs by the process.

We observe the trace from the environment's perspective since we observe events from the environment's perspective.

If we write down a completed session (a communication sequence after which the environment neither sends nor receives any signals), we call this trace a failure trace[1].

Example The failure trace $a?b!$ means that the environment sent a signal through channel a to the process, after which the environment received a signal through channel b, after which nothing occurred. (So the environment did not receive or send any signals anymore).

If we consider the set of all possible sessions, (the set of all failure traces), we call this set the failure set. A failure set is a precise description of the communication capabilities of a process, and we consider it to be the definition of a process.

2.3.2 Operational view of failure sets

We can regard failure sets as the rules for a game between the process and its environment. The purpose of the game is to reach a failure trace together.

There is always a current trace (say cu), that is initially empty (ϵ). Both the process and the environment are allowed to extend cu by attaching signals to the end.

the following rules hold:

- The trace cu always has to be a prefix of a trace in the failure set of the process.
- The environment is allowed to attach inputs only.
- The process is allowed to attach outputs only.
- Since a communication session has to end at a failure trace, the process is obliged to extend the trace cu if it is not in the failure set. (All possible finished sessions are in the failure set, and hence when cu is not in the failure set, the session was not finished.)

example 1: Let $\{\epsilon, a?b!\}$ be a failure set. The process is initially not obliged to communicate, and isn't even able to do so, as the only non-empty trace starts with $a?$ and only the environment can send inputs. The environment is not allowed to send any input other than $a?$. When the environment sends a signal through a (attaches a? to the current trace), the process has to sent a signal through b eventually, as $a?$ is not a trace in the failure set.

Example 2: Let $\{b!\}$ be a failure set. The process has to sent a $b!$, since ϵ is not in the failure set.

[1] This name is traditional [1]. The process is allowed to *fail* to send outputs after this communication sequence

3 Delay insensitivity and receptiveness

3.1 introduction

In our model processes are *receptive*[6]. This means that the environment is always allowed to send a signal to the process,, and the process cannot stop the environment from doing so. This might look obvious, but is not the case in most synchronous models, like CSP[5] for example.

In our model processes are ***delay-insensitive***. This means that we will make no assumptions on the speed of signals traveling through channels. If two signals travel through two different channels, we will make no assumptions on their relative speeds.

The delay insensitivity gives rise to a robustness requirement; if two signals travel through the same channel at the same time, interference might occur. (For example: Two voltage changes on a wire, or when one inserts two coins in a coffee machine in rapid succession, or when the coffee machine would pour two coffees at the same time). When two signals travel through the same channel at the same time during the communication with a process we say that that process is in an undesirable (or chaotic) state. Of course, in practice it is possible that nothing would go wrong (the signals do not necessarily interfere), however, Murphy's Law teaches us to expect the worst. The fact that in our model, the possibility of undesirable behaviour is considered undesirable, is called robustness. Our model is a pessimistic model.

3.2 Failure sets

From now on, we identify failure sets with delay-insensitive receptive processes.

We shall give the restrictions that a set of traces has to meet to be a failure set. These restrictions follow from the informal description of delay-insensitive processes that we gave earlier.

In order to formalize failure sets a number of auxiliary functions of type $\mathcal{P}(A^*) \rightarrow \mathcal{P}(A^*)$ are introduced. ($\mathcal{P}$ is the powerset function, and A^* the set of all finite traces over A).

$$prefs.S = \{s, t : st \in S : s\}$$

This is the well known prefix closure of a set of traces.

$$\mathbf{t}.S = \{t : t \in O^* \wedge st \in S : s\}$$

Operationally, we can view $\mathbf{t}.S$, where S is a failure set (or equivalently a process), as the set of all traces that a process S can extend (by producing outputs, or doing nothing) to a trace in set S.

$$\mathbf{d}.S = \{t, b : b \in O \wedge tbb \in \mathbf{t}.S : t\} \cup \{t, a : a \in I \wedge taa \in \mathbf{t}.S : taa\}$$

Operationally, if S is a process, and $x \in \mathbf{d}.S$, communicating x with S will put S in an undesirable state. A process that can produce two outputs on the same channel is already in an undesirable state.

Finally we introduce the *reordering* relation on traces. The fact that we know nothing about the relative speed with which signals travel through channels (and the fact that signals on different channels may overtake each other), is modeled in the reordering closure demands on failure sets.

Suppose the current trace in a communication session has just been extended by $a!b!$. This means the environment received an $a!$ followed by a $b!$. Since we make no assumption on the relative speeds of signals, the environment cannot know whether the process sent an $a!$ followed by a $b!$, or a $b!$ followed by an $a!$. Therefore, all subsequent communications after receiving $a!b!$ should also be possible after receiving $b!a!$.

Furthermore, if the environment sent an $a?$ followed by a $b?$, it cannot be sure that the process received the signals in that particular order. Therefore, all subsequent communications after sending $a?b?$ should also be possible after sending $b?a?$.

Finally, suppose the current trace has the shape $sb!a?$. This means that the trace s was communicated, after which the environment received a $b!$, after which the environment sent an $a?$. The environment could also impatiently send an $a?$ before the $b!$ arrived. Therefore, $sa?b!$ is also a possible current trace. Furthermore, the environment can not be certain whether the $b!$ was a result of the process receiving an $a?$, or was already sent when the environment sent an $a?$. Thus all subsequent communications after communicating $sb!a?$ are possible after communicating $sa?b!$. The reverse is not the case. If a current trace has the shape $sa?b!$, it could be the case that the process sending an $b!$ can only be a direct result of the process receiving an $a?$ (causality), in which case communicating $sb!a?$ is not even possible.

$$s \preceq^1 t \equiv (\exists x, a, b, y : a \in I \vee b \in O : s = xaby \wedge t = xbay)$$

The single 'swapping' of signals as described above is modeled in the relation $\preceq^1$. Since a failure trace that was the result of another failure trace might gave rise to yet more failure traces, we define the reordering relation to be $\preceq$, the transitive reflexive closure of $\preceq^1$.
We denote the space of failure sets with $\mathbf{F}$, and the space of vectors with size n of failure sets with $\mathbf{F}^n$.

The restrictions that a failure set $F \in \mathbf{F}$ has to meet are:

1. $\mathbf{t}.F \neq \emptyset$
2. $prefs.(\mathbf{t}.F) \subseteq \mathbf{t}.F$
3. $(\mathbf{d}.F)A^* \subseteq \mathbf{d}.F$
4. $(\mathbf{t}.F)I \subseteq \mathbf{t}.F$
5. $t \in F \wedge s \preceq t \Rightarrow s \in F$

We will give a pointwise motivation for these restrictions.

1. We demand this so communication is possible. If there exists no trace that the process is able to extend to a failure trace, there is no failure trace, and no communication is possible, as the current trace has to be a prefix of a failure trace.
2. This restriction makes sure that the environment is not obliged to extend a trace to a failure trace. Suppose a trace $s \in prefs.(\mathbf{t}.F)$ is not in the set $\mathbf{t}.F$. s is a prefix of a failure trace (see definition t), so s is a possible current trace. Furthermore, the trace s does not occur in $\mathbf{t}.F$, so the process cannot extend s to a failure trace. This means that only the environment can extend the trace s to a failure trace, and hence that we would require signals from the environment, which is not our intention.
3. This restriction makes sure that a process in an undesirable state will remain in an undesirable state. Murphy's Law again.
4. This restriction allows the environment to send signals to the process at any time. Again we don't want to restrict the environment in any way. This is the receptiveness demand that we mentioned earlier. How the process will react to a signal from the environment is described by the failure set.
5. This restriction was already elaborated upon at the introduction of the reordering relation.

With these restrictions, the space of failure sets (and hence processes) is precisely defined. We will give some simple lemmas, the proof of which we will omit.

Let F a failure set.

$$prefs.F = \mathbf{t}.F \tag{1}$$

Which we shall refer to as "prefs lemma".

Let s, t traces, let $x \in A$.

$$s \preceq xt \equiv (\exists s_0, s_1 : s_0 \in I^* \vee x \in O : s = s_0 x s_1 \wedge s_0 s_1 \preceq t) \tag{2}$$

Which we shall refer to as "reordering lemma".

For a more elaborate description of failure sets and their properties we refer to [7].

4 A metric for failure sets

4.1 Introduction

A metric is a distance function on a certain space, in our case the space of failure sets. We could say that our metric is "a function that denotes how much failure sets differ". We assume that the processes that we compare have equal alphabets.

If we consider how processes P and Q, differ initially, we can distinguish between two possibilities: (recall that processes are identified with failure sets).

- The process P is not obliged to send a signal ($\epsilon \in P$), but the process Q is ($\epsilon \notin Q$). Or vice versa.
- The process P can send a signal that the process Q cannot send . Or vice versa.

Of course, symmetrically, we could try the distinction: 'The process P can receive a signal that the process Q cannot receive, or vice versa', but since our processes are receptive, this cannot occur.

The distinction that we make initially can also be made after communicating a certain trace. We consider the shortest trace that the environment can communicate with the two processes after which the two processes differ as a measure for the way in which processes differ [4]. To this effect we use the *min* quantor and infix-operator and ∞ as a neutral element. Furthermore, we use $|x|$ to denote the length of trace x,

This leads us to the following expression for the length of the shortest trace after which two processes P and Q may behave differently.

$$disc(P,Q) = min \begin{pmatrix} (Min\ x : x \in P \not\equiv x \in Q : |x|) \\ (Min\ x, b : xb \in prefs.P \not\equiv xb \in prefs.Q \wedge b \in O : |x|) \end{pmatrix}$$

The first quantification denotes the length of the shortest trace after which the first kind of difference occurs. The second quantification denotes the length of the shortest trace after which the second kind of difference occurs.

Since we have $prefs.P = t.P$ and $(t.P)I \subseteq t.P$, and $P \neq \emptyset$, we can rewrite the expression $disc(P,Q)$ to:

$$(Min\ x : x \in P \not\equiv x \in Q : |x|) min (Min\ x : x \in prefs.P \not\equiv x \in prefs.Q : |x| - 1)$$

The smaller the trace after which two processes differ, the sooner we can distinguish between them. We define the function ρ as

$$\rho(P,Q) = 2^{-disc(P,Q)}$$

Since we allowed ∞ as a neutral element, we see that the image of *disc* is given by the set:

$$\{k : k \in \mathbb{N} : k\} \cup \{\infty\}$$

Likewise, the image of ρ is given by the set:

$$\{k : k \in \mathbb{N} : 2^{-k}\} \cup \{0\}$$

One useful property of ρ is the metric lemma:

$$\rho(P,Q) \leq 2^{-k}$$
$$\equiv$$
$$(\forall x : |x| < k : x \in P \equiv x \in Q) \wedge (\forall x : |x| < k + 1 : x \in prefs.P \equiv x \in prefs.Q)$$

Proof:

$$\rho(P,Q) \leq 2^{-k}$$

$\equiv$ { definition ρ, property 2^x }

$(Min\ x : x \in P \not\equiv x \in Q : |x|) min(Min\ x : x \in prefs.P \not\equiv x \in prefs.Q : |x| - 1) \geq k$

$\equiv$ { property min }

$(\forall x : x \in P \not\equiv x \in Q : |x| \geq k) \wedge (\forall x : x \in prefs.P \not\equiv x \in prefs.Q : |x| - 1 \geq k)$

$\equiv$ { trading }

$(\forall x : |x| < k : x \in P = x \in Q) \wedge (\forall x : |x| < k + 1 : x \in prefs.P = x \in prefs.Q)$

4.2 ρ is an ultra metric

To prove that ρ is an ultra metric, we have to show the following four properties.

$$\begin{array}{lll} \rho(P,Q) & \geq & 0 \\ \rho(P,Q) & = & \rho(Q,P) \\ \rho(P,Q) = 0 & \equiv & P = Q \\ \rho(P,Q) & \leq & max(\rho(P,R), \rho(R,Q)) \end{array}$$

The first three properties are easy to see. The last property is trivial if we choose $R = P$ or $R = Q$. In the other case we observe for any k:

$\rho(P,Q) \leq 2^{-k}$

$\equiv$ { Metric lemma }

$(\forall x : |x| < k : x \in P \equiv x \in Q) \wedge (\forall x : |x| < k + 1 : x \in prefs.P \equiv x \in prefs.Q)$

$\Leftarrow$ { transitivity $=$ }

$(\forall x : |x| < k : x \in P \equiv x \in R) \wedge (\forall x : |x| < k + 1 : x \in prefs.P \equiv x \in prefs.R) \wedge$
$(\forall x : |x| < k : x \in R \equiv x \in Q) \wedge (\forall x : |x| < k + 1 : x \in prefs.R \equiv x \in prefs.Q)$

$\equiv$ { metric lemma }

$\rho(P,R) \leq 2^{-k} \wedge \rho(R,Q) \leq 2^{-k}$

$\equiv$ { definition max }

$max(\rho(P,R), \rho(R,Q)) \leq 2^{-k}$

Now, if $P \not\equiv R$ or $R \not\equiv Q$, we choose k such that we have equality in the last line, and this establishes our proof obligation.

4.3 Metrics in vector space

If $P, Q \in \mathbf{F}^n$, we define:

$$\rho_n(P,Q) = (Max\ i : 0 \leq i < n : \rho(P_i, Q_i))$$

and

$$disc_n(P,Q) = (Min\ i : 0 \leq i < n : disc(P_i, Q_i))$$

It can easily be proven that ρ_n is also an ultra metric. We define ρ on vectors as ρ_n. Likewise we define $disc$ on vectors as $disc_n$. It is easy to observe that

$$\rho(P,Q) = 2^{-disc(P,Q)}$$

still holds.

5 The DI-algebra

5.1 introduction

The di-algebra was developed as a means to specify and compare delay-insensitive processes [6]. The process expressions in the di-algebra are constructed from 1 basic process and a number of operators. The space $\mathbf{F}$ that was introduced previously is used as a semantic space for the process expressions[7]. Or in other words, process expressions denote failure sets.

We will identify expressions with their meaning, and make no distinction between syntactic and semantic space. The definition of the operators and the basic process $\perp$ is given in the appendix.

Our discussion of process expressions and the di-algebra is not complete, but should provide the reader with a working knowledge. Process expressions are extensively discussed in [7].

- $\perp \in \mathbf{F}$,
- $[i : i \in I : a_i \rightarrow P_i] \in \mathbf{F}$, if $a_i \in A$, and $P_i \in \mathbf{F}$, for any $i \in I$,
- $(\sqcap j : j \in J : P_j) \in \mathbf{F}$, if $P_j \in \mathbf{F}$ for any $j \in J$,
- $P/a \in \mathbf{F}$, if $P \in \mathbf{F}$ and $a \in A$, and
- $P||Q \in \mathbf{F}$, if $P, Q \in \mathbf{F}$,

,where I is a finite set of indices, and J is a non-empty finite set of indices.

The parallel composition $||$ will not be investigated in this work, since its metric properties are hard to describe, and we do not need it for our proof methods rule, as will be clear from the example in chapter 7. The only property of $||$ we will use is that $P||Q$ is a failure set if P and Q are failure sets.

We will give an informal operational description of the processes associated with process expressions.

- $\perp$ is the undesired process.
- $[i :: a_i \rightarrow A_i]$ is the guarded choice. If a guard a_i belongs to the set of output channels ($a_i \in O$), the process may send an output signal through a_i, and subsequently behave as the process A_i. If a guard a_i belongs to the set of input channels, ($a_i \in I$), and a signal through channel a_i is received by the process, the process may subsequently behave as process A_i. If output guards occur, and the environment does not send a signal corresponding with a guard, the process will send an output eventually.
- $(\sqcap i :: A_i)$ is the nondeterministic choice. The process will nondeterministically behave as one of the processes A_i.
- P/a, is pronounced 'P after a'. If a is an input channel, the process will behave like P after the environment has sent a signal through channel a. If a is an output channel, the process will behave like P after the environment has received a signal through channel a. Note that this last case is sometimes undefined, as a process might not be able to send a signal through channel a.

5.2 Guardedness and contraction

A function f is contracting if and only if

$$(\exists c : 0 \leq c < 1 : (\forall x, y :: \rho(f.x, f.y) \leq c\rho(x, y))) \tag{3}$$

We would like to derive which functions that are composed of constructor functions are contracting. If a function f is contracting, $f \circ f$ is even more contracting (the constant for $f \circ f$ equals at least c^2). Therefore, it might be interesting to derive a lower bound for the constant c, or derive a measurement for the contractiveness of functions.

Let f be a function of type $\mathbf{F}^n \rightarrow \mathbf{F}$. We would like the value k to be a measurement of the contractiveness of the function f, such that:

$$(\forall x, y :: \rho(f.x, f.y) \leq 2^{-k}\rho(x, y))$$

First we observe:

$$
\begin{array}{ll}
 & (\forall x, y :: \rho(f.x, f.y) \le 2^{-k}\rho(x, y)) \\
\equiv & \{ \text{ term holds for } x = y \ \} \\
 & (\forall x, y : x \ne y : \rho(f.x, f.y) \le 2^{-k}\rho(x, y)) \\
\equiv & \{ \text{ definition } \rho \ \} \\
 & (\forall x, y : x \ne y : 2^{-disc(f.x, f.y)} \le 2^{-k}2^{-disc(x,y)}) \\
\equiv & \{ \ 2^x \text{ monotone in } x \ \} \\
 & (\forall x, y : x \ne y : -disc(f.x, f.y) \le -k - disc(x, y)) \\
\equiv & \{ \text{ calculus, } x \ne y \Rightarrow disc(x, y) \in \mathbb{N} \ \} \\
 & (\forall x, y : x \ne y : k \le disc(f.x, f.y) - disc(x, y)) \\
\equiv & \{ \text{ definition } Min \ \} \\
 & k \le (Min\ x, y : x \ne y : disc(f.x, f.y) - disc(x, y))
\end{array}
$$

We define the function *igds* (input guardedness) as

$$igds.f = (Min\ x, y : x \ne y : disc(f.x, f.y) - disc(x, y)) \tag{4}$$

It is obvious that from $igds.f > 0$ it follows that f is contracting. We named the function 'input guardedness', since its properties are closely connected to the intuitive notion of input guardedness;i.e. the number of input guards preceding variables in a functional abstraction of a process expression.

The function *igds* has the following properties:

$$
\begin{array}{lcll}
igds.c & = & \infty & \text{, if } c \text{ is a constant function.} \\
igds.id & = & 0 & \text{, if } id \text{ is the identity function.} \\
igds.(\sqcap i :: f_i) & \ge & (Min\ i :: igds.f_i) & \\
igds.(g/a) & \ge & (igds.g) - 1 & \\
igds.[i :: a_i \rightarrow f_i] & \ge & min \begin{pmatrix} (Min\ i : a_i \in I : igds.f_i + 1) \\ (Min\ i : a_i \in O : igds.f_i) \end{pmatrix} & \\
igds.F & = & (Min\ i :: igds.f_i) & \text{, if } F \text{ is a vector valued function} \\
igds.f & \ge & 0 & \text{, if } f \text{ is a projection (i.e. } f.X = X_i)
\end{array}
$$

Each property mentioned above follows from a property of *disc*. Some of these properties are not trivial, and those are proven in the appendix.

$$
\begin{array}{lcl}
disc(P, P) & = & \infty \\
disc((\sqcap i :: P_i), (\sqcap i :: Q_i)) & \ge & (Min\ i :: disc(P_i, Q_i)) \\
disc(P/a, Q/a) & \ge & disc(P, Q) - 1 \\
disc([i :: a_i \rightarrow P_i], [i :: a_i \rightarrow Q_i]) & \ge & min \begin{pmatrix} (Min\ i : a_i \in I : disc(P_i, Q_i) + 1) \\ (Min\ i : a_i \in O : disc(P_i, Q_i)) \end{pmatrix}
\end{array}
$$

Note that *igds* for vector functions follows directly from the definition of *disc* for vector functions. Also *igds* for projections can easily be derived by *igds* for identity and *igds* for vector valued functions.

We say that a function f is *input guarded* if $igds.f > 0$.

5.3 fixpoints and recursive equations

Ordinarily when we write equations of the form $x = f.x$, we define x to be a fixpoint of the function f. We use $\mu\ x.(f.x)$ to denote the smallest fixpoint of f.

Likewise, in the di-algebra we use $P = F.P$ to denote $P = \mu\ x.(F.x)$, where P can be a simple name as well as a vector. There are several requirements on these equations for the processes to be well defined. One of these demands is that there are no variables in $F.P$ that do not occur in P.

Examples of valid recursive equations in the di-algebra are:

$W = [a? \rightarrow [b! \rightarrow W]]$

$P = [b! \rightarrow Q]$
$Q = [a? \rightarrow P]$

5.4 uniqueness of fixpoints

A contracting function has at most one fixpoint.

(let $p = f.p$ and $q = f.q$)

$$
\begin{array}{ll}
 & \text{true} \\
\equiv & \{\ f \text{ contracting }\} \\
 & (\exists c : 0 \leq c < 1 : (\forall x, y :: \rho(f.x, f.y) \leq c\rho(x, y))) \\
\Rightarrow & \{\ \text{Instantiation }\} \\
 & (\exists c : 0 \leq c < 1 : \rho(f.p, f.q) \leq c.\rho(p, q)) \\
\equiv & \{\ p \text{ and } q \text{ are fixpoints }\} \\
 & (\exists c : 0 \leq c < 1 : \rho(p, q) \leq c.\rho(p, q)) \\
\equiv & \{\ \text{calculus }\} \\
 & (\exists c : 0 \leq c < 1 : (1 - c)\rho(p, q) \leq 0) \\
\equiv & \{\ \text{calculus } (1 - c) \geq 0\ \} \\
 & \rho(p, q) \leq 0 \\
\equiv & \{\ \rho \text{ is a metric function }\} \\
 & \rho(p, q) = 0 \\
\equiv & \{\ \rho \text{ is a metric function }\} \\
 & p = q
\end{array}
$$

So we have:

$$igds.f > 0 \wedge p = f.p \wedge q = f.q \Rightarrow p = q$$

Not all functions derived from constructor functions have a unique fixpoint. If we consider the function F, defined by $F.P = [a? \rightarrow P]/a?$, we see that both $\perp$ and $a? \rightarrow \perp$ are fixpoints of this function. However, all functions have a smallest fixpoint, as proven in [7], and since contracting functions have at most one fixpoint, input guarded functions have a unique fixpoint. This means that we can prove that two processes (or vectors of processes) are equal by proving that they are both a fixpoint of the same input guarded function.

6 A contraction based proof method

6.1 introduction

As we saw earlier, input guarded functions are contracting with regard to their arguments. We would like to use this property to construct proofs of the following shape:

$$
\begin{array}{ll}
 & X \\
= & \{\ \text{argument why } X = F.X\ \} \\
 & F.X \\
= & \{\ F \text{ is input guarded }\} \\
 & F.Y \\
= & \{\ \text{argument why } Y = F.Y\ \} \\
 & Y
\end{array}
$$

Unfortunately, the fact that F is input guarded does not imply that $F.X = F.Y$. However, from the contractiveness of F we can conclude: $\rho(F.X, F.Y) \leq c.\rho(X,Y)$, where $0 \leq c < 1$. This suggests that instead of looking at equality, it might be more appropriate to look at the distance between the processes that occur in a proof. Since ρ is an ultra metric, the distance between the first and last process in such a proof would be less or equal to the maximum distance occurring. We will formalize this notion in the following section.

6.2 A new equivalence relation

6.2.1 definition

If we consider failure sets (or vectors of failure sets) x and y we can define a new relation $=_{xy}$ on failure sets or vectors of failure sets as:

$$r =_{xy} s \equiv \rho(r,s) < \rho(x,y) \vee r = s$$

Theorem: The relation $=_{xy}$ is an equivalence relation.

Proof: We have to prove that $=_{xy}$ is symmetric, reflexive and transitive. The first and second property are easily derived from the definition of $=_{xy}$. For the transitivity of $=_{xy}$ we observe:

$$\begin{array}{ll}
 & r =_{xy} s \wedge s =_{xy} t \\
\equiv & \quad \{ \text{ definition } =_{xy} \ \} \\
 & (\rho(r,s) < \rho(x,y) \vee r = s) \wedge (\rho(s,t) < \rho(x,y) \vee s = t) \\
\equiv & \quad \{ \text{ distribution } \} \\
 & (\rho(r,s) < \rho(x,y) \wedge \rho(s,t) < \rho(x,y)) \vee (\rho(r,s) < \rho(x,y) \wedge s = t) \vee \\
 & (\rho(s,t) < \rho(x,y) \wedge r = s) \vee (r = s \wedge s = t) \\
\Rightarrow & \quad \{ \text{ calculus, Leibnitz } \} \\
 & \max((\rho(r,s), \rho(s,t))) < \rho(x,y) \vee \rho(r,t) < \rho(x,y) \vee \rho(r,t) < \rho(x,y) \vee r = t \\
\Rightarrow & \quad \{ \ \rho \text{ is an ultra metric, idempotency of } \vee \ \} \\
 & \rho(r,t) < \rho(x,y) \vee r = t \\
\equiv & \quad \{ \text{ definition } \} \\
 & r =_{xy} t
\end{array}$$

6.2.2 properties

The relation $=_{xy}$ has a number of useful properties. We will list some of them below.

- $$x =_{xy} y \equiv x = y \qquad (5)$$

  The proof of (5) follows directly from the definition of $=_{xy}$.

- If A and B are vectors of the same size (say n) then we have:

  $$A =_{xy} B \equiv (\forall i : 0 \leq i < n : A_i =_{xy} B_i) \qquad (6)$$

  Note that x and y can also be vectors.

- One very useful property of $=_{xy}$ is the following.

  $$\rho(r,s) \leq \rho(x,y) \wedge igds.f > 0 \Rightarrow f.r =_{xy} f.s \qquad (7)$$

  This can easily be derived from the definition of $=_{xy}$, and $igds.f > 0 \Rightarrow f$ contracting. Since we have for vectors A, B:

  $$\rho(A_i, B_i) \leq \rho(A,B)$$

  we conclude from (7)

  $$igds.f > 0 \Rightarrow f.A_i =_{AB} f.B_i \qquad (8)$$

6.3 Contraction based proof methods

6.3.1 A method to prove equality

We can summarize the results of 6.2 in the following theorem:
theorem: Let X and Y be vectors of failure sets. Let $\approx$ be the relation $=_{XY}$. Then we have:

- $X \approx Y \Rightarrow X = Y$, from (5)
- $(\forall i :: X_i \approx Y_i) \equiv X \approx Y$, from (6)
- $\approx$ is an equivalence relation.
- $igds.f > 0 \Rightarrow f.X_i \approx f.Y_i$ (contraction), from (8)

We are now able to incorporate the contraction rule in our proofs, as was our intention. If we consider this chapter's introduction, we could prove $X = Y$ in the following manner:

Let $\approx$ be the equivalence relation $=_{XY}$.

$$\begin{array}{ll} & X \\ = & \{ \text{ Argument why } X = F.X \ \} \\ & F.X \\ \approx & \{ \ F \text{ is input guarded, contraction } \} \\ & F.Y \\ = & \{ \text{ Argument why } Y = F.Y \ \} \\ & Y \end{array}$$

So we have $X \approx Y$, and it follows that $X = Y$.

6.3.2 A method to prove refinement

The refinement relation in the di-algebra is defined as

$$p \sqsubseteq q \equiv p = p \sqcap q$$

We define refinement on vectors X, Y as

$$X \sqsubseteq q \equiv (\forall i :: X_i \sqsubseteq Y_i)$$

We will present a contraction based proof method to prove refinement. For the derivation of this method, and the definition of the relation $\sqsubseteq_{xy}$ we refer to the appendix.
theorem: Let $X, Y \in F^N$. Suppose we want to prove $X \sqsubseteq Y$. Then we have:

- $(\forall i :: X_i \sqsubseteq_{XY} Y_i) \Rightarrow X \sqsubseteq Y$. (25,26)
- $igds.f > 0 \wedge f$distributes over $\cup \Rightarrow f.X_i \sqsubseteq_{XY} f.Y_i$. (29)
- $\sqsubseteq_{XY}$ is transitive. (31)
- $p \sqsubseteq q \Rightarrow p \sqsubseteq_{XY} q$. (32)

In [3] it was shown that all operators in the di-algebra distribute over $\cup$.

7 Example

7.1 introduction

In the early nineties, Udding designed an asynchronous buffer to show the power of using parallel composition and di-algebra in design. Udding used an intuitive approach he called induction, that we can now formalize by using our proof method. We will repeat a formalized version of Udding's proof here to give an example of the use and power of the proof method, as well as show the power of using di-algebra in design.

7.2 syntax and rewrite rules

Two new constructions appear: *skip* guards and prefixing (;). We did not mention these earlier, as they can always be rewritten to one of the constructs we did introduce, as was shown in [3]. ($a; f$ is an abbreviation of $[a \rightarrow f]$), so $igds.(a?; f) \geq 1 + igds.f$.

A list of di-algebra rewrite rules that are used is given in the appendix The complete description of all rewrite rules, and the proof that they hold **F**, can be found in [7].

7.3 definitions and intentions

We define the vector function $F : \mathbf{F}^{n+1} \rightarrow F^{n+1}$, where $n > 0$, by the equation:

$$\begin{array}{rcl} (F.X)_0 & = & a?b!; X_n \\ (F.X)_i & = & [a?{\rightarrow}b!; X_{i+1} \;\square\; p?{\rightarrow}r!; X_{i-1}], \text{ where } 0 < i < n \\ (F.X)_n & = & p; ?r!; X_{n-1} \end{array} \tag{9}$$

Obviously, we can regard the right hand occurrences of X_i as a projection applied to X.
First we observe for F:

$$\begin{array}{ll} & igds.F \\ = & \quad \{ \text{ property } igds \ \} \\ & (Min\ i :: igds.F_i) \end{array}$$

And for the component functions of F we observe:

Let $i = 0$

$$\begin{array}{ll} & igds.F_i \\ = & \quad \{ \ i = 0 \ \} \\ & igds.a?; b!; X_n \\ \geq & \quad \{ \text{ property } igds \ \} \\ & 1 + igds.b!; X_n \\ \geq & \quad \{ \text{ property } igds \ \} \\ & 1 + igds.X_n \\ \geq & \quad \{ \ igds \text{ for projection } \} \\ & 1 \end{array}$$

Let $0 < i < n$

$$\begin{array}{ll} & igds.F_i \\ = & \quad \{ \text{ definition } F \ \} \\ & igds.[a?{\rightarrow}b!; X_{i+1} \;\square\; p?{\rightarrow}r!; X_{i-1}] \\ \geq & \quad \{ \text{ property } igds \ \} \\ & (1 + igds.b!; X_{i+1}) min (1 + igds.r!; X_{i-1}) \\ \geq & \quad \{ \text{ distribution of } + \text{ over } min, \text{ property } igds, \ \} \\ & 1 + (igds.X_{i+1}\ min\ igds.X_{i-1}) \\ \geq & \quad \{ \ igds \text{ for projection } \} \\ & 1 \end{array}$$

Let $i = n$

$$\begin{array}{ll} & igds.F_i \\ = & \quad \{ \text{ definition } F \ \} \end{array}$$

$$\begin{array}{ll} & igds.p?; r!; X_{n-1} \\ \geq & \quad \{ \text{ property } igds \ \} \\ & 1 + r!; X_{n-1} \\ \geq & \quad \{ \ igds \text{ for guarded choice and projection } \} \\ & 1 \end{array}$$

So we have $igds.F_i > 0$ for all i, and hence $igds.F > 0$ [2]. From $igds.F > 0$ and F continuous as proven in [7], it follows that F has an unique fixpoint, which we name B.

$$B = F.B$$

Operationally we can think of B_i as a buffer that can contain as much as n elements, and currently contains i elements. We can regard $a?$ as a store action, $b!$ as a confirmation of a store action, $p?$ as a retrieve action, and $r!$ as a confirmation of a retrieve action.
We define the function G as

$$G.X = b?; k?; a!; l!; X$$

G is an input guarded function, and we define Ct as its unique fixpoint.

$$Ct = G.Ct$$

We will show that the parallel composition of $Ct/b?$ and B_o results in an $n+1$ buffer, with a and b replaced by k and l respectively. Therefore we define:

$$\begin{array}{rcl} (H.X)_0 & = & a?b!; X_n \\ (H.X)_i & = & [a? \rightarrow b!; X_{i+1} \ \Box \ p? \rightarrow r!; X_{i-1}], \text{ where } 0 < i < n+1 \\ (H.X)_{n+1} & = & p; ?r!; X_n \end{array} \tag{10}$$

The function H is obviously input guarded, and we define D as its unique fixpoint.

$$D = H.D$$

So our proof obligation is

$$B_0 || (Ct/b?) = D_0$$

7.4 proof

First we will prove the following equalities:

$$\begin{array}{lcll} B_i/a? & = & b!; B_{i+1} & \text{for } 0 \leq i < n-1 \\ B_{n-1}/a? & = & [a? \rightarrow \perp \Box \ skip \rightarrow b!; B_n] & \\ B_n/a? & = & [a? \rightarrow \perp \Box \ p? \rightarrow r!; (B_{n-1}/a?)] & \\ B_i/p? & = & r!; B_{i-1} & \text{for } 1 < i \leq n \\ B_1/p? & = & [p? \rightarrow \perp \Box \ skip \rightarrow r!; B_0] & \\ B_0/p? & = & [p? \rightarrow \perp \Box \ a? \rightarrow b!; (B_1/p?)] & \end{array}$$

We can regard the left hand side (L) and the right hand side (R) of these equations as vectors of $2n+2$ failure sets. Of course proving $L = R$ is equivalent to proving all equalities.

If we apply the proof method of chapter 5, we have:

[2]From now on we will not explicitly prove input guardedness, since it is easy to observe

Let L, R be the vectors introduced above. Let $\approx$ be the equivalence relation $=_{LR}$. Then, in order to prove $L = R$, it is enough to prove $(\forall i :: L_i \approx R_i)$, and we have for all functions $\mathbb{F} \to \mathbb{F}$:

$$igds.f > 0 \Rightarrow (\forall i :: f.L_i = f.R_i),$$

which we shall refer to as contraction.

We are now ready to prove the $2n+2$ equalities. Di-algebra rewrite rules that are used can be found in the appendix.

proof:
First of all, we observe from the definition of an n-place buffer, using after through output-prefixing and through guarded choice, and two outputs in a row that for $0 \leq i < n$:

$$(b!; B_i)/a? = \perp = (r!; B_{i+1})/p? \tag{11}$$

For $0 = i \leq n-1$ we derive:

$B_0/a?$
$=$ { one choice is no choice and after through input-prefixing }
$[a? \to \perp \ \square\ skip \to b!; B_1]$
$=$ { (11), using that $0 \leq n-1$ }
$[a? \to (b!; B_1)/a? \ \square\ skip \to b!; B_1]$
$=$ { input extension, and one choice is no choice }
$b!; B_1$

By symmetry we now also have $B_n/p? = r!; B_{n-1}$, for $1 < 1 = n$.

For $0 < i < n-1$ we derive:

$B_i/a?$
$=$ { after through guarded choice and output prefixing }
$[a? \to \perp \ \square\ skip \to b!; B_{i+1} \ \square\ p? \to r!; (B_{i-1}/a?)]$
$\approx$ { contraction, left hand side occurring in input guarded function }
$[a? \to \perp \ \square\ skip \to b!; B_{i+1} \ \square\ p? \to r!; b!; B_i]$
$=$ { output reordering }
$[a? \to \perp \ \square\ skip \to b!; B_{i+1} \ \square\ p? \to b!; r!; B_i]$
$\approx$ { contraction, right hand side occurring in input guarded function }
$[a? \to \perp \ \square\ skip \to b!; B_{i+1} \ \square\ p? \to b!; (B_{i+1}/p?)]$
$=$ { 11 }
$[a? \to (b!; B_{i+1})/a? \ \square\ skip \to b!; Bi+1 \ \square\ p? \to b!; (B_{i+1})/p?]$
$=$ { after through output, input extension and one choice is no choice }
$b!; B_{i+1}$

By symmetry we now also have $B_i/p? \approx r!; B_{i-1}$ for $1 < i < n$.

For $B_{n-1}/a?$ and $n = 1$ the result follows immediately from the definition and after through guarded choice.

For $n > 1$ we derive:

$B_{n-1}/a?$
$=$ { after through guarded choice and output prefixing }
$[a? \to \perp \ \square\ skip \to b!; B_n \ \square\ p? \to r!; (B_{n-2}/a?)]$
$\approx$ { contraction }

$[a? \rightarrow \perp \;\Box\; skip \rightarrow b!; B_n \;\Box\; p? \rightarrow r!; b!; B_{n-1}]$

$\approx$ { output reordering and contraction }

$[a? \rightarrow \perp \;\Box\; skip \rightarrow b!; B_n \;\Box\; p? \rightarrow b!; (B_n/p?)]$

$=$ { one choice is no choice, after through output-prefixing, and absorption }

$[a? \rightarrow \perp \;\Box\; skip \rightarrow b!; B_n \;\Box\; p? \rightarrow [a? \rightarrow \perp \;\Box\; skip \rightarrow (b!; B_n)/p?]]$

$=$ { after through guarded choice }

$[a? \rightarrow \perp \;\Box\; skip \rightarrow b!; B_n \;\Box\; p? \rightarrow [a? \rightarrow \perp \;\Box\; skip \rightarrow b!; B_n]/p?]$

$=$ { input extension }

$[a? \rightarrow \perp \;\Box\; skip \rightarrow b!; B_n]$

Symmetrically we now have $B_1/p? \approx [p? \rightarrow \perp \;\Box\; skip \rightarrow r!; B_0]$

The final two equations for $B_n/a?$ and $B_0/p?$ follow directly from the definition, one choice is no choice, and after through guarded choice.

Now we have proven $L_i \approx R_i$ for all possible i and hence $L = R$. Or in other words that all equalities hold.

For the two-output C-element we derive:

$Ct/b?$

$=$ { One choice is no choice, and after through guarded choice }

$[b? \rightarrow \perp \;\Box\; skip \rightarrow k?; a!; l!; Ct]$

$=$ { one choice is no choice and *skip* elimination }

$[b? \rightarrow \perp \;\Box\; k? \rightarrow a!; l!; Ct]$

We are now ready to prove the composition of $Ct/b?||B_0$ to yield D_0. We will prove this by proving simultaneously:

$$Ct/b?||B_i = D_i^{n+1}, \text{for all } (0 \leq i < n)$$

And again we can regard these equalities as a vector equality, so let L be the left hand side vector, and let R be the right hand side vector, we define $\approx$ as $=_{LR}$.

We observe for $i = 0$:

$Ct/b?||B_0$

$=$ { parallel composition through guarded choice }

$[k? \rightarrow (a!; l!; Ct)||B_0]$

$=$ { parallel composition through output prefixing }

$k?; l!; (Ct||B_0/a?)$

$=$ { previous equalities }

$k?; l!; (Ct||b!; B_1)$

$=$ { $b!; B_1$ can be rewritten to $[a? \rightarrow \perp \;\Box\; skip \rightarrow b!; B_1]$ using input extension }

$k?; l!; (Ct||[a? \rightarrow \perp \;\Box\; skip \rightarrow b!; B_1])$

$=$ { parallel composition through guarded choice, one choice is no choice }

$k?; l!; (Ct||(b!; B_1))$

$\approx$ { Parallel composition through output prefixing, contraction }

$k?l!; D_1^{n+1}$

$=$ { definition D_0^{n+1} }

D_i^{n+1}

For $0 < i < n$ we derive:

$Ct/b?||B_i$

$=$ { parallel composition. through guarded choice }

$[k?\rightarrow(a!; l!; Ct)||B_i \;\square\; p?\rightarrow Ct/b?||(r!; B_{i-1})]$

$=$ { Parallel composition. through output prefixing }

$[k?\rightarrow l!; (Ct||B_i/a?) \;\square\; p?\rightarrow r!; (Ct/b?||B_{i-1})]$

$\approx$ { contraction }

$[k?\rightarrow l!; (Ct||B_i/a?) \;\square\; p?\rightarrow r!; D^{n+1}_{i-1}]$

$=$ { previous equalities, using that $b!; B_{i+1}$ can be rewritten to $[a?\rightarrow bot \;\square\; skip\rightarrow b!; B_{i+1}]$ with input extension }

$[k?\rightarrow l!; (Ct||[a?\rightarrow\bot \;\square\; skip\rightarrow b!; B_{i+1}]) \;\square\; p?\rightarrow r!; D^{n+1}_{i-1}]$

$=$ { parallel composition through guarded choice, one choice is no choice }

$[k?\rightarrow l!; (Ct||(b!; B_{i+1})) \;\square\; p?\rightarrow r!; D^{n+1}_{i-1}]$

$=$ { parallel composition through output prefixing }

$[k?\rightarrow l!; (Ct/b?||B_{i+1}) \;\square\; p?\rightarrow r!; D^{n+1}_{i-1}]$

$\approx$ { contraction }

$[k?\rightarrow l!; D^{n+1}_{i+1} \;\square\; p?\rightarrow r!; B^{n+1}_{i-1}]$

$=$ { definition D }

D^{n+1}_i

And finally we derive:

$Ct/b?||B_n$

$=$ { parallel composition through guarded choice, one choice is no choice }

$[k?\rightarrow((a!; l!; Ct)||B_n) \;\square\; p?\rightarrow(Ct/b?||r!; B_{n-1})]$

$=$ { parallel composition through output prefixing }

$[k?\rightarrow l!; (Ct||B_n/a?) \;\square\; p?\rightarrow r!; (Ct/b?||B_{n-1})]$

$\approx$ { contraction }

$[k?\rightarrow l!; (Ct||B_n/a?) \;\square\; p?\rightarrow r!; D^{n+1}_{n-1}]$

$=$ { previous equalities }

$[k?\rightarrow l!; (Ct||[a?\rightarrow\bot \;\square\; p?\rightarrow r!; (B_{n-1}/a?)] \;\square\; p?\rightarrow r!; D^{n+1}_{n-1}]$

$=$ { parallel composition through guarded choice }

$[k?\rightarrow l!; [p?\rightarrow(Ct||r!; (B_{n-1}/a?)] \;\square\; p?\rightarrow r!; D^{n+1}_{n-1}]$

$=$ { parallel composition through output prefixing }

$[k?\rightarrow l!; p?; r!; (Ct||B_{n-1}/a?) \;\square\; p?\rightarrow r!; Dn + 1_{n-1}]$

$=$ { previous equalities }

$[k?\rightarrow l!; p?; r!; (Ct||[a?\bot \;\square\; skip\rightarrow b!; B_n]) \;\square\; p?\rightarrow r!; D^{n+1}_{n-1}]$

$=$ { unfolding Ct, parallel composition through guarded choice }

$[k?\rightarrow l!; p?; r!; (Ct||b!; B_n) \;\square\; p?\rightarrow r!; D^{n+1}_{n-1}]$

$=$ { parallel composition through output prefixing }

$[k?\rightarrow l!; p?; r!; (Ct/b?||B_n) \;\square\; p?\rightarrow r!; D^{n+1}_{n-1}]$

$\approx$ { contraction }

$[k?\rightarrow l!; p?; r!; (D^{n+1}_n) \;\square\; p?\rightarrow r!; Dn + 1_{n-1}]$

$=$ { definition D^{n+1}_{n+1} }

$[k?\rightarrow l!; D^{n+1}_{n+1} \;\square\; p?\rightarrow r!; D^{n+1}_{n-1}]$

$=$ { definition D }

D^{n+1}_n

So now we have proven $L =_{LR} R$, where L_i was defined as $Ct/b?||B_i$, and R_i as D_i. It follows that $L = R$, and hence $Ct/b?||B_0 = D_0$, which was our goal.

8 Conclusions and further research

We have defined a metric on processes. We have introduced two relations ($=_{xy}$ and $\sqsubseteq_{xy}$) on processes and vectors of processes. We have derived proof methods that use these relations to prove equality and refinement respectively. The proof methods we presented validate most proofs in the delay insensitive algebra that used "induction".

We have shown that input guarded functions have at most one fixpoint. It is easy to see that output guarded functions have at most one fixpoint, since they will be rewritable to $\bot$ (using two outputs in a row) when unfolded twice. Therefore, all guarded functions have at most one fixpoint.

We have derived properties of *igds* that allow us to formally derive the input guardedness of a function on processes or vectors of processes. We have given an example of such a derivation.

It is possible to define our metric function ρ using syntactical (as opposed to our semantical failure sets) constructions only. If this is done, the metric function and its properties might be presented as an axiom, thus not only specifying equality in our di-algebra, but also inequality.

The proof methods we presented are as of yet not incorporated in the algebraic proof assistant [2], but are currently being implemented.

9 Acknowledgements

I would like to thank Jan Tijmen Udding for his supervision, and helping me get rid of unimportant details, Rutger Dijkstra for his suggestions and comments that lead to a "cleaner" approach, and Paul Lucassen for listening and commenting upon my ideas. I would also like to thank the members of the reading club: Rutger Dijkstra, Wim Hesselink, Jan Jongejan, Jan Eppo Jonker and Jan Tijmen Udding for suggestions, ideas, and introducing me into the world of formal computing science. Finally, I would like to thank Ineke Kruizinga for coffee in the early mornings, and Rix Groenboom for tales from Spain.

References

[1] S. Brookes, C. Hoare, and A. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(7):560–599, 1984.

[2] R. Groenboom, C. Hendriks, I. Polak, J.T.Udding, C. Hendriks, and J. Terlouw. Algebraic proof assistants in hol. In B. Móller, editor, *Mathematics of Program Construction*, LNCS 947, pages 304–321. Springer-Verlag, 1995.

[3] R. Groenboom, M. B. Josephs, P. G. Lucassen, and J. T. Udding. Normal form in a delay-insensitive algebra. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 57–70. Elsevier Science Publishers, 1993.

[4] M. Hennessy. An algebraic theory of fair asynchronous communicating processes. *Theoretical Computer Science*, 49:121–143, 1987.

[5] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, Aug. 1978.

[6] M. B. Josephs and J. T. Udding. An overview of DI algebra. In *Proc. Hawaii International Conf. System Sciences*, volume I. IEEE Computer Society Press, Jan. 1993.

[7] P. G. Lucassen. *A Denotational Model and Composition Theorems for a Calculus of Delay-Insensitive Specifications*. PhD thesis, Dept. of C.S., Univ. of Groningen, The Netherlands, May 1994.

[8] A. Peeters. The Asynchronous Bibliography. Available for anonymous ftp on Internet. Uniform Resource Locator (URL) `ftp://ftp.win.tue.nl/pub/tex/async.bib.Z`. Corresponding e-mail address: `async-bib@win.tue.nl`.

[9] T. Pratchett. *The colour of magic*. Corgi Books, 1990.

A Definition of operators

In [7] the definition of operators in the di-algebra was given terms of failure sets. We will repeat that definition here for all processes not composed with $||$, as we need it in the proofs to follow. As we can see in the buffer section, it is not necessary to include any aspect of parallel composition, except that the parallel composition of two failure sets yields a failure set.

$$\begin{array}{lcll}
\perp & = & A^* & \\
(\sqcap i :: A_i) & = & (\cup i :: A_i) & \\
[i :: a_i \rightarrow A_i] & = & \{y, i, z : y \in A_i \wedge z \preceq a_i y : z\} \cup & \\
 & & \{y, s, i : ya_i \in \mathrm{t}.A_i \wedge a_i \in O : ys\} \cup & \\
 & & SD \ \cup QS\ (i :: a_i) & \\
P/a & = & \{y : ay \in P : y\} & \text{if } P/a \text{ is not empty, and undefined otherwise}
\end{array}$$

Where the constant SD and the function QS are defined as:

$$SD = \{y, a, s : a \in I \wedge y \preceq aas : y\}$$

$$QS(i :: a_i) = \begin{array}{ll} (I - \{i :: a_i\})^* & \text{if } (\forall i :: a_i \in I) \\ \emptyset & \text{otherwise} \end{array}$$

B The *disc* proofs

B.1 introduction

In this appendix we prove two of the properties of *disc* that were stated earlier. The reader should be able to verify the other properties for herself.

First we observe for the *disc* function:

Let $P, Q \in \mathbf{F}$, let $k \in \mathbb{N} \cup \{\infty\}$ then we have:

$$disc(P, Q) \geq k \equiv (\forall x : |x| < k : x \in P \equiv x \in Q) \wedge (\forall x : |x| < k+1 : x \in prefs.P \equiv x \in prefs.Q),$$

which we shall refer to as '*disc* lemma'. The proof of this property is easily derived from the metric lemma.

B.2 proof

We start with the proof of

$$disc((\sqcap i :: A_i), (\sqcap i :: B_i)) \geq (Min\ i :: disc(A_i, B_i)) \tag{12}$$

In order to prove (12), it is enough to prove
For any $k \in \mathbb{N} \cup \infty$ we have:

$$(Min\ i :: disc(A_i, B_i)) \geq k \Rightarrow disc((\sqcap i :: A_i), (\sqcap i :: B_i)) \geq k \tag{13}$$

As instantiating the antecedent of (13) with equality, leads directly to (12).
In order to prove 13, we observe for the antecedent:

$$\begin{array}{ll}
 & (Min\ i :: disc(A_i, B_i)) \geq k) \\
\equiv & \quad \{ \text{ property } Min \ \} \\
 & (\forall i :: disc(A_i, B_i) \geq k) \\
\equiv & \quad \{ \ disc \text{ lemma } \} \\
 & (\forall i :: (\forall z : |z| < k : z \in A_i \equiv z \in B_i) \wedge (\forall z : |z| < k+1 : z \in prefs.A_i \equiv z \in prefs.B_i))
\end{array}$$

So if we observe the consequent for a fixed k, we have for any i, and any x:

$$|x| < k \Rightarrow x \in A_i \equiv x \in B_i \qquad (*)$$
$$|x| < k+1 \Rightarrow x \in prefs.A_i \equiv x \in prefs.B_i \qquad (**)$$

Now we derive for the consequent:

$disc((\sqcap i :: A_i), (\sqcap i :: B_i)) \geq k$
$\equiv$ { *disc* lemma }
$(\forall z : |z| < k : z \in (\sqcap i :: A_i) \equiv z \in (\sqcap i :: B_i)) \wedge$
$(\forall z : |z| < k+1 : z \in prefs.(\sqcap i :: A_i) \equiv z \in prefs.(\sqcap i :: B_i))$

and we observe for any z such that $|z| < k$:

$z \in (\sqcap i :: A_i)$
$\equiv$ { $\sqcap = \cup$ }
$z \in (\cup i :: A_i)$
$\equiv$ { definition $\cup$ }
$(\exists i :: z \in A_i)$
$\equiv$ { $(*)$ }
$(\exists i :: z \in B_i)$
$\equiv$ { same way around }
$z \in (\sqcap i :: B_i)$

And likewise for any z such that $|z| < k+1$:

$z \in prefs.(\sqcap i :: A_i)$
$\equiv$ { $\sqcap = \cup$ }
$z \in prefs.(\cup i :: A_i)$
$\equiv$ { *prefs* distributes over $\cup$,definition $\cup$ }
$(\exists i :: z \in prefs.A_i)$
$\equiv$ { $(**)$ }
$(\exists i :: z \in prefs.B_i)$
$\equiv$ { same way around }
$z \in prefs.((\sqcap i :: B_i)$

,which concludes our proof.

We now prove the property of *disc* with regard to the guarded choice. This is the hardest proof of all, as the definition of the guarded choice is fairly complicated.
Our proof obligation is:

$$disc([i :: a_i \rightarrow A_i], [i :: a_i \rightarrow B_i]) \quad \geq \quad min \left(\begin{array}{l} (Min\ i : a_i \in I : disc(A_i, B_i) + 1) \\ (Min\ i : a_i \in O : disc(A_i, B_i)) \end{array} \right) \qquad (14)$$

As we saw earlier, in order to prove (14), it is enough to prove for any $k \in \mathbb{N} \cup \{\infty\}$ that

$$(Min\ i : a_i \in I : disc(A_i, B_i) + 1) min (Min\ i : a_i \in O : disc(A_i, B_i)) \geq k \qquad (15)$$

implies

$$disc.([i :: a_i \rightarrow A_i], [i :: a_i \rightarrow B_i]) \geq k \qquad (16)$$

First, we observe for (15)

$(Min\ i : a_i \in I : disc(A_i, B_i) + 1) min (Min\ i : a_i \in O : disc(A_i, B_i) \geq k$
$\equiv$ { property *min* }

$(\forall i : a_i \in I : disc(A_i, B_i) \geq k-1) \wedge (\forall i : a_i \in O : disc(A_i, B_i) \geq k)$

$\equiv$ { *disc* lemma }

$(\forall i : a_i \in I : (\forall z : |z| < k-1 : z \in A_i \equiv z \in B_i) \wedge$
$\quad (\forall z : |z| < k : z \in prefs.A_i \equiv z \in prefs.B_i)) \wedge$
$(\forall i : a_i \in O : (\forall z : |z| < k : z \in A_i \equiv z \in B_i) \wedge$
$\quad (\forall z : |z| < k+1 : z \in prefs.A_i \equiv z \in prefs.B_i))$

So if we observe (16) for a fixed k, we have for any i and z:

$$a_i \in I \wedge |z| < k-1 \quad \Rightarrow \quad z \in A_i \equiv z \in B_i \qquad (17)$$
$$a_i \in I \wedge |z| < k \quad \Rightarrow \quad z \in prefs.A_i \equiv z \in prefs.B_i \qquad (18)$$
$$a_i \in O \wedge |z| < k \quad \Rightarrow \quad z \in A_i \equiv z \in B_i \qquad (19)$$
$$a_i \in O \wedge |z| < k+1 \quad \Rightarrow \quad z \in prefs.A_i \equiv z \in prefs.B_i \qquad (20)$$

Now we rewrite 16 to a conjunction, and then derive each conjunct from the antecedent.

$disc.([i :: a_i \rightarrow A_i], [i :: a_i \rightarrow B_i]) \geq k$

$\equiv$ { *disc* lemma }

$(\forall z : |z| < k : z \in [i :: a_i \rightarrow A_i] \equiv z \in [i :: a_i \rightarrow B_i]) \wedge$
$(\forall z : |z| < k+1 : z \in prefs.[i :: a_i \rightarrow A_i] \equiv z \in prefs.[i :: a_i \rightarrow B_i])$

$\Leftarrow$ { definition guarded choice, *prefs* distributes over $\cup$, Leibnitz }

$(A)(\forall z : |z| < k : z \in \{r, v, i : r \in A_i \wedge v \preceq a_i r : v\} \equiv z \in \{r, v, i : r \in B_i \wedge v \preceq a_i r : v\}) \wedge$
$(B)(\forall z : |z| < k :$
$\quad z \in \{r, s, i : ra_i \in \mathbf{t}.A_i \wedge a_i \in O : rs\} \equiv z \in \{r, s, i : ra_i \in \mathbf{t}.B_i \wedge a_i \in O : rs\} \wedge$
$(C)(\forall z : |z| \leq k+1 :$
$\quad z \in prefs.\{r, v, i : r \in A_i \wedge v \preceq a_i r : v\} \equiv z \in prefs.\{r, v, i : r \in B_i \wedge v \preceq a_i r : v\}) \wedge$
$(D)(\forall z : |z| \leq k+1 :$
$z \in prefs.\{r, s, i : ra_i \in \mathbf{t}.A_i \wedge a_i \in O : rs\} \equiv z \in prefs.\{r, s, i : ra_i \in \mathbf{t}.B_i \wedge a_i \in O : rs\}$

In every case we will show that the equivalence holds for a fixed i. According to Leibnitz, the theorem will be a consequence.

Proof of A: Let z be any trace that satisfies $|z| < k$, Let i be an index.

$z \in \{r, v : r \in A_i \wedge v \preceq a_i r : v\}$

$\equiv$ { definition $\{..\}$ }

$(\exists r, v : r \in A_i \wedge v \preceq a_i r : z = v)$

$\equiv$ { trading, one point }

$(\exists r : z \preceq a_i r : r \in A_i)$

$\equiv$ { $z \preceq a_i r \Rightarrow |r| < |z|$, thus $|r| < k-1$, (17) or (19) }

$(\exists r : z \preceq a_i r : r \in B_i)$

$\equiv$ { same way around }

$z \in \{r, v : r \in B_i \wedge v \preceq a_i r : v\}$

Proof of B: Let z be any trace satisfying $|z| < k$, let i be any index, if $a_i \in I$ the sets are trivially equal, so we only consider an i such that $a_i \in O$.

$z \in \{r, s : ra_i \in \mathbf{t}.A_i \wedge a_i \in O : rs\}$

$\equiv$ { definition $\{..\}$, trading, $a_i \in O$ }

$(\exists r, s : z = rs : ra_i \in \mathbf{t}.A_i)$

$\equiv$ { (1) }

$(\exists r, s : z = rs : ra_i \in prefs.A_i)$

$\equiv$ $\{\ z = rs \Rightarrow |ra_i| \leq |z| + 1 < k + 1,\ (20)\ \}$

$(\exists r, s : z = rs : ra_i \in \mathit{prefs}.B_i))$

$\equiv$ $\{$ same way around $\}$

$z \in \{r, s : ra_i \in \mathrm{t}.B_i \wedge a_i \in O : rs\}$

The proof of C is the most complicated. We shall distinguish between $a_i \in O$ and $a_i \in I$.
Proof of C (a) : Let $a_i \in I$, and $|z| < k + 1$

$z \in \mathit{prefs}.\{r, v : r \in A_i \wedge v \preceq a_i r : v\}$

$\equiv$ $\{$ definitions prefs,$\{..\}$ $\}$

$(\exists u, r, v : r \in A_i \wedge v \preceq a_i r : zu = v)$

$\equiv$ $\{$ nesting, one point $\}$

$(\exists u, r : r \in A_i : zu \preceq a_i r)$

$\equiv$ $\{$ reordering lemma, $a_i \in I$ $\}$

$(\exists u, r : r \in A_i : (\exists s_0, s_1 : s_0 \in I^* : zu = s_0 a_i s_1 \wedge s_0 s_1 \preceq r))$

$\equiv$ $\{$ nesting , trading $\}$

$(\exists u, s_0, s_1 : s_0 \in I^* \wedge zu = s_0 a_i s_1 : (\exists r : r \in A_i : s_0 s_1 \preceq r))$

$\equiv$ $\{$ A_i is a failure set, hence closed with regard to $\preceq$ $\}$

$(\exists u, s_0, s_1 : s_0 \in I^* \wedge zu = s_0 a_i s_1 : s_0 s_1 \in A_i)$

$\equiv$ $\{$ case distinction $\}$

$(\exists u, s_0, s_1 : zu = s_0 a_i s_1 \wedge s_0 \in I^* \wedge (z \in I^* \vee z \notin I^*) : s_0 s_1 \in A_i)$

$\equiv$ $\{$ distribution , splitting the domain $\}$

$(\exists u, s_0, s_1 : zu = s_0 a_i s_1 \wedge s_0 \in I^* \wedge z \in I^* : s_0 s_1 \in A_i) \vee$
$(\exists u, s_0, s_1 : zu = s_0 a_i s_1 \wedge s_0 \in I^* \wedge z \notin I^* : s_0 s_1 \in A_i)$

$\equiv$ $\{$ $(\Leftarrow)$ choose $s_0, s_1, u := z, a_i a_i, a_i a_i a_i$ (NB $s_0 s_1 \in SD \subseteq A_i$) $\}$

$z \in I^* \vee (\exists u, s_0, s_1 : zu = s_0 a_i s_1 \wedge s_0 \in I^* \wedge z \notin I^* : s_0 s_1 \in A_i)$

$\equiv$ $\{$ structure of s_1 $\}$

$z \in I^* \vee (\exists u, s_0, s_1, w, m : z = s_0 a_i w \wedge u = m \wedge s_1 = wm \wedge z \notin I^* \wedge s_0 \in I^* : s_0 s_1 \in A_i)$

$\equiv$ $\{$ substitution, single point $\}$

$z \in I^* \vee (\exists s_0, w, m : z = s_0 a_i w \wedge z \notin I^* \wedge s_0 \in I^* : s_0 wm \in A_i)$

$\equiv$ $\{$ nesting, definition prefs $\}$

$z \in I^* \vee (\exists s_0, w : z = s_0 a_i w \wedge z \notin I^* \wedge s_0 \in I^* : s_0 w \in \mathit{prefs}.A_i)$

$\equiv$ $\{$ $|s_0 w| < |z|$, (18) $\}$

$z \in I^* \vee (\exists s_0, w : z = s_0 a_i w \wedge z \notin I^* \wedge s_0 \in I^* : s_0 w \in \mathit{prefs}.B_i)$

$\equiv$ $\{$ same way around $\}$

$z \in \mathit{prefs}.\{r, v : r \in B_i \wedge v \preceq a_i r : v\}$

Proof of C (b) : Let $a_i \in O$, en $|z| < k + 1$

$z \in \mathit{prefs}.\{r, v : r \in A_i \wedge v \preceq a_i r : v\}$

$\equiv$ $\{$ definitions prefs,$\{..\}$ $\}$

$(\exists u, r, v : r \in A_i \wedge v \preceq a_i r : zu = v)$

$\equiv$ $\{$ nesting, single point $\}$

$(\exists u, r : r \in A_i : zu \preceq a_i r)$

$\equiv$ $\{$ reordering lemma, $a_i \in O$ $\}$

$(\exists u, s_0, s_1 : zu = s_0 a_i s_1 : s_0 s_1 \in A_i)$

$\equiv$ $\{$ structure of z $\}$

$(\exists u, s_0, s_1 : s_0 s_1 \in A_i :$
$\quad (\exists w, m : z = w \wedge u = m a_i s_1 : s_0 = wm) \vee$

$(\exists w, m : z = s_0 a_i w \wedge u = m : s_1 = wm))$

$\equiv$ { distribution $\vee$ over $\exists$,nesting,trading }

$(\exists w, m, u, s_0, s_1 : s_0 s_1 \in A_i : z = w \wedge u = m a_i s_1 \wedge s_0 = wm) \vee$

$(\exists w, m, u, s_0, s_1 : s_0 s_1 \in A_i : z = s_0 a_i w \wedge u = m \wedge s_1 = wm)$

$\equiv$ { substitution }

$(\exists w, m, u, s_0, s_1 : zms_1 \in A_i : z = w \wedge u = m a_i s_1 \wedge s_0 = wm) \vee$

$(\exists w, m, u, s_0, s_1 : s_0 wm \in A_i : z = s_0 a_i w \wedge u = m \wedge s1 = wm)$

$\equiv$ { single point }

$(\exists m, s_1 :: zms_1 \in A_i) \vee (\exists s_0, w, m : s_0 wm \in A_i : z = s_0 a_i w)$

$\equiv$ { definition $prefs$, renaming }

$z \in prefs.A_i \vee (\exists k, l : z = k a_i l : kl \in prefs.A_i)$

$\equiv$ { $(|kl| < |z|)$, (20) twice }

$z \in prefs.B_i \vee (\exists k, l : z = k a_i l : kl \in prefs.B_i)$

$\equiv$ { same way around }

$z \in prefs.\{r, v : r \in B_i \wedge v \preceq a_i r : v\}$

And finally we have to prove D. We consider any z such that $|z| < k + 1$. We consider only any i such that $a_i \in O$. (The equivalence is trivial for $a_i \in I$).

$z \in prefs.\{r, s : r a_i \in \mathbf{t}.A_i \wedge a_i \in O : rs\}$

$\equiv$ { definition $\{..\}$, single point,$a_i \in O$ }

$(\exists u, r, s : r a_i \in \mathbf{t}.A_i : zu = rs))$

$\equiv$ { $prefs$ lemma }

$(\exists u, r, s : zu = rs : r a_i \in prefs.A_i)$

$\equiv$ { distribution }

$(\exists u, r, s : |z| \leq |r| : zu = rs \wedge r a_i \in prefs.A_i) \vee$

$(\exists u, r, s : |z| > |r| : zu = rs \wedge r a_i \in prefs.A_i)$

$\equiv$ { structure of z }

$(\exists v, r, u, s : zv = r : zu = rs \wedge r a_i \in prefs.A_i) \vee$

$(\exists v, r, u, s : z = rv \wedge v \in A^+ : zu = rs \wedge r a_i \in prefs.A_i)$

$\equiv$ { substitution }

$(\exists v, r, u, s : zv = r : zu = rs \wedge z v a_i \in prefs.A_i) \vee$

$(\exists v, r, u, s : z = rv \wedge v \in A^+ : rvu = rs \wedge r a_i \in prefs.A_i)$

$\equiv$ { $(\Leftarrow)$ choose $v, r, u, s := a?a?, za?a?, a?a?, \epsilon$, then $za?a?a_i \in A_i$ }

$z \in prefs.A_i \vee$

$(\exists v, r, u, s : z = rv \wedge v \in A^+ : rvu = rs \wedge r a_i \in prefs.A_i)$

$\equiv$ { single point }

$z \in prefs.A_i \vee$

$(\exists v, r : z = rv \wedge v \in A^+ : r a_i \in prefs.A_i)$

$\equiv$ { $|r| < z \Rightarrow |rb| \leq |z|$, (20) }

$z \in prefs.B_i \vee$

$(\exists v, r : z = rv \wedge v \in A^+ : r a_i \in prefs.B_i)$

$\equiv$ { same way around }

$z \in prefs.\{r, s : r a_i \in \mathbf{t}.B_i \wedge a_i \in O : rs\}$

And this concludes our proof.

C Derivation of a proof method for refinement

C.1 Preliminaries

We would like the same sort of contraction based rule for proving implementation ($\sqsubseteq$), as we have for proving equality. The relation $p \sqsubseteq q$ is defined as $p = p \sqcap q$, for failure sets p and q.

First we define for $X, Y \in \mathbf{F}^n$:

$$X \sqsubseteq Y \equiv (\forall i :: X_i \sqsubseteq Y_i) \tag{21}$$

and likewise

$$(X \sqcap Y)_i = X_i \sqcap Y_i \tag{22}$$

C.2 The new relation

We define

$$p \sqsubseteq_{xy} q \equiv p =_{x,x\sqcap y} p \sqcap q \tag{23}$$

where $=_{x,x\sqcap y}$ is the earlier introduced equivalence relation. Note that x and y can be vectors. Note that $\sqsubseteq_{xy}$ for vectors is defined , since $=_{x,x\sqcap y}$ is defined for vectors.

C.3 relation to $\sqsubseteq$

Let $x, y \in \mathbf{F}$. Then we have:

$$x \sqsubseteq_{xy} y \equiv x \sqsubseteq y \tag{24}$$

proof:

$x \sqsubseteq_{xy} y$
$\equiv$ { definition $\sqsubseteq_{xy}$ }
$x =_{x,x\sqcap y} x \sqcap y$
$\equiv$ { property $=_{x,x\sqcap y}$ }
$x = x \sqcap y$
$\equiv$ { definition $\sqsubseteq$ }
$x \sqsubseteq y$

Furthermore, we observe for vectors $X, Y \in \mathbf{F}^n$.

$$X \sqsubseteq_{XY} Y \equiv (\forall i : X_i \sqsubseteq_{XY} Y_i) \tag{25}$$

$X \sqsubseteq_{XY} Y$
$\equiv$ { Definition $\sqsubseteq_{XY}$ }
$X =_{X,X\sqcap Y} X \sqcap Y$
$\equiv$ { property $=_{X,X\sqcap Y}$ }
$(\forall i :: X_i =_{X,X\sqcap Y} X_i \sqcap Y_i)$

So we have

$$X \sqsubseteq_{XY} Y \equiv X \sqsubseteq Y \tag{26}$$

C.4 Effect of application of input guarded functions

$$\rho(r, r \sqcap s) \le \rho(x, x \sqcap y) \wedge igds.f > 0 \wedge f \text{distributes over } \cup \Rightarrow f.r \sqsubseteq_{xy} f.s \tag{27}$$

proof:

$f.r \sqsubseteq_{xy} f.s$
$\equiv$ { definition $\sqsubseteq_{xy}$ }
$f.r =_{x,x \sqcap y} f.r \sqcap f.s$
$\equiv$ { f distributes over $\cup$ }
$f.r =_{x,x \sqcap y} f.(r \sqcap s)$
$\Leftarrow$ { property $=_{x,x \sqcap y}$, f input guarded }
true

And since we have for vectors X and Y:

$$\rho(X_i, X_i \sqcap Y_i) \le \rho(X, X \sqcap Y) \tag{28}$$

We observe:

$$igds.f > 0 \wedge f \text{ continuous} \Rightarrow f.X_i \sqsubseteq_{XY} f.Y_i \tag{29}$$

C.5 Properties of $\sqsubseteq_{xy}$

The following two properties describe transitive behaviour of the relation $\sqsubseteq_{xy}$

$$p \sqsubseteq_{xy} q \wedge q \sqsubseteq r \quad \Rightarrow \quad p \sqsubseteq_{xy} r \tag{30}$$
$$p \sqsubseteq_{xy} q \wedge q \sqsubseteq_{xy} r \quad \Rightarrow \quad p \sqsubseteq_{xy} r \tag{31}$$

proof of (31):

$p \sqsubseteq_{xy} q \wedge q \sqsubseteq_{xy} r$
$\equiv$ { definition $\sqsubseteq_{xy}$ }
$p =_{x,x \sqcap y} q \sqcap p \wedge q =_{x,x \sqcap y} r \sqcap q$
$\Rightarrow$ { $f.x = c \sqcap x$ is not expanding, definition $=_{x,x \sqcap y}$ }
$p =_{x,x \sqcap y} q \sqcap p \wedge p \sqcap q =_{x,x \sqcap y} r \sqcap q \sqcap p$
$\Rightarrow$ { $=_{x,x \sqcap y}$ is transitive }
$p =_{x,x \sqcap y} r \sqcap q \sqcap p$
$\Rightarrow$ { property $\sqsubseteq$ }
$p =_{x,x \sqcap y} r \sqcap q \sqcap p \wedge r \sqcap q \sqcap p \sqsubseteq p \sqcap r$
$\Rightarrow$ { (30) }
$p =_{x,x \sqcap y} p \sqcap r$
$\equiv$ { definition $\sqsubseteq_{xy}$ }
$p \sqsubseteq_{xy} r$

proof of (30)

$p \sqsubseteq_{xy} q \wedge q \sqsubseteq r$
$\equiv$ { definition $\sqsubseteq_{xy}$ }
$p =_{x,x \sqcap y} p \sqcap q \wedge q \sqsubseteq r$
$\equiv$ { definition $=_{x,x \sqcap y}$ }
$(\exists c : 0 \le c < 1 : \rho(p, p \sqcap q) \le c\rho(x, x \sqcap y)) \wedge q \sqsubseteq r$
$\equiv$ { distribution, definition $\sqsubseteq$ and $\sqcap$ }
$(\exists c : 0 \le c < 1 : \rho(p, p \cup q) \le c\rho(x, x \sqcap y) \wedge r \subseteq q)$

$\Rightarrow$ $\{\ a \subseteq b \subseteq c \Rightarrow \rho(a,b) \leq \rho(a,c)\ (*)$, with $a,b,c := p, p \cup r, p \cup q\ \}$

$(\exists c : 0 \leq c < 1 : \rho(p, p \cup r) \leq c\rho(x, x \sqcap y))$

$\equiv$ $\{$ definition $=_{x,x\sqcap y}$ and $\sqsubseteq_{xy}$ $\}$

$p \sqsubseteq_{xy} r$

And finally we prove $(*)$:

$\rho(a,b) \leq \rho(a,c)$

$\equiv$ $\{$ definition ρ $\}$

$disc(a,b) \geq disc(a,c)$

and

$disc(a,b)$

$=$ $\{$ definition $disc$ $\}$

$(Min\ x : x \in a \not\equiv x \in b : |x|) min (Min\ x : x \in prefs.a \not\equiv x \in prefs.b : |x| - 1)$

$=$ $\{\ a \subseteq b\ \}$

$(Min\ x : x \notin a \wedge x \in b : |x|) min (Min\ x : x \notin prefs.a \wedge x \in prefs.b : |x| - 1)$

$\geq$ $\{\ b \subseteq c\ \}$

$(Min\ x : x \notin a \wedge x \in c : |x|) min (Min\ x : x \notin prefs.a \wedge x \in prefs.c : |x| - 1)$

$=$ $\{\ a \subseteq c\ \}$

$(Min\ x : x \in a \not\equiv x \in c : |x|) min (Min\ x : x \in prefs.a \not\equiv x \in prefs.c)$

$=$ $\{$ definition $disc$ $\}$

$disc(a,c)$

Furthermore, we have for any x, y:

$$p \sqsubseteq q \Rightarrow p \sqsubseteq_{xy} q \tag{32}$$

proof:

$p \sqsubseteq q$

$\equiv$ $\{$ definition $\sqsubseteq$ $\}$

$p = p \sqcap q$

$\Rightarrow$ $\{\ =_{x,x\sqcap y}$ is reflexive $\}$

$p =_{x,x\sqcap y} p \sqcap q$

$\equiv$ $\{$ definition $\}$

$p \sqsubseteq_{xy} q$

C.6 New proof method

We collect the results of the previous sections in the following proof method:

Let $X, Y \in F^N$. Suppose we want to prove $X \sqsubseteq Y$. Then we can use the following theorems:

1. $(\forall i :: X_i \sqsubseteq_{XY} Y_i) \Rightarrow X \sqsubseteq Y$. (25,26)
2. $igds.f > 0 \wedge f$distributes over $\cup \Rightarrow f.X_i \sqsubseteq_{XY} f.Y_i$. (29)
3. $\sqsubseteq_{XY}$ is transitive. (31)
4. $p \sqsubseteq q \Rightarrow p \sqsubseteq_{XY} q$. (32)

Thus 1 gives us a proof obligation, and 3,4 allows us to create linear derivations. 2 allows us to use contractive properties of operators in the di-algebra, as all operators in the di-algebra distribute over $\cup$.

C.7 abstract example

We could use the proof method to create proofs of the following shape:
Suppose, we want to prove that $X_i \sqsubseteq Y_i$ for all i.

X_i
$\sqsubseteq$ { di-algebra argument }
$f.X_i$
$\sqsubseteq_{XY}$ { f input guarded and distributes over $\cup$ }
$f.Y_i$
$\sqsubseteq$ { di-algebra argument }
Y_i

Therefore, $X_i \sqsubseteq_{XY} Y_i$ for all i and hence $X_i \sqsubseteq Y_i$ for all i.

D Some di-algebra rewrite rules

- Output reordering

$$c!; d!; P = d!; c!; P$$

- Two outputs in a row

$$c!; [skip \rightarrow c!; P \;\square\; S] = \bot$$

- One choice is no choice

$$[skip \rightarrow P] = P \text{ and } [a? \rightarrow P] = a?; P$$

- Input-extension

$$[S] = [a? \rightarrow [S]/a? \;\square\; b? \rightarrow [S]/b? \;\square\; S]$$

- Absorption

$$[a? \rightarrow \bot \;\square\; b? \rightarrow [a? \rightarrow P \;\square\; S_0] \;\square\; S_1] = [a? \rightarrow \bot \;\square\; b? \rightarrow [S_0] \;\square\; S_1]$$

- skip-elimination

$$[a? \rightarrow \bot \;\square\; skip \rightarrow [S]] = [a? \rightarrow \bot \;\square\; S]$$

- After through guarded choice

$$[S]/a? = [a? \rightarrow \bot \;\square\; S']$$

Where S' is formed by substituting for each alternative $A \in S$ the new alternative $A/a?$ defined by

$$\begin{array}{lcl} (skip \rightarrow P)/a? & = & skip \rightarrow (P/a?) \\ (a? \rightarrow P)/a? & = & skip \rightarrow P \\ (b \rightarrow P)/a? & = & b \rightarrow (P/a?). \text{ for } b \neq a? \end{array}$$

- Parallel composition through guarded choice

$$[S_0]||[S_1] = [S]$$

Where S is formed from the alternatives in S_0 and S_1 in the following way. For each alternative in S_0 of the form $skip \rightarrow P$, we have $skip \rightarrow (P||[S_1])$ in S. For each alternative in S_0 of the form $a? \rightarrow P$, with a not in the output alphabet of $[S_1]$, we have $a? \rightarrow (P||[S_1])$ in S. The alternatives in S_1 contribute to the alternatives in S_0 in a similar way.

- Parallel composition through output-prefixing

$$(c!;P)||Q = \left(\begin{array}{ll} \mathrm{P||(Q/c?)} & \text{if } c \text{ in the input alphabet of } Q \\ \mathrm{c!;(P||Q)} & \text{otherwise} \end{array} \right)$$