An Approach to Process Algebra using IMPS

F. Javier Thayer *

April 11, 1995

Abstract

In this paper we develop a formal mathematical model for processes, along the lines of Communicating Sequential Processes (CSP). Our development is completely general and is capable of dealing with time. Rather than building semantic models on a traditional trace-based failures model, we replace the set of traces by the set of elements of a monoid. In the case of untimed CSP, this monoid is taken to be the set of all event sequences under concatenation; mathematically this is the free monoid generated by the event alphabet. For timed CSP, one natural monoid that may be considered is the free monoid generated by the set of atomic events together with the wait operators $W(t)$ for t a real number with the relations $W(t+s) = W(t)W(s)$.

*Supported by the United States Army CECOM under contract DAAB07-94-C-H601, through MITRE's Technology Program. Author's address: The MITRE Corporation, 202 Burlington Rd, Bedford MA 01730-1420 USA; Telephone: 617-271-2749; Fax: 617-271-3816; E-mail: jt@mitre.org

Contents

1 Introduction

In this paper we develop a new approach to process algebra and CSP. One goal of the paper is to provide a foundation for subsequent work in process specification, and to this end, the development has been done entirely using the IMPS system. For an overview of the IMPS system, see [6, 7]. Moreover, we have departed from the usual formulations in which external behaviors are modeled as sequences of events called traces. In the following presentation, we have adopted a mathematically more satisfying and more general approach which replaces the set of traces by the set of elements of a monoid.

We have two main examples in mind:

1. In the case of untimed CSP, this monoid is taken to be the set of all event sequences under concatenation; mathematically, this is the free monoid generated by the event alphabet.

2. For timed CSP, one natural monoid that may be considered is the free monoid generated by the set of atomic events together with the wait operators $W(t)$ for t a real number with the relations $W(t+s) = W(t)W(s)$.

In this approach to process semantics, in general there may be no "atomic" events, that is, events which are not themselves non-trivial products. Indeed, actions have to be defined as certain classes of monoid elements under an equivalence relation $\cong$, where $a \cong b$ for monoid elements a, b means that either

- Both a and b are the identity of the monoid, or
- a, b have a nontrivial common prefix

The set of actions consist of "germs" of monoid elements. If we think of monoid elements as being the external interactions of a process, the germ of an element is a mathematical representation of the process's initial behavior. For example, in the case of untimed CSP, the germ is the first action of the process.

In order to relate this approach to the myriad of existing approaches to process theory, we have developed a theory of mathematical structure which captures some of the formalism of process transition. We refer to this structure as a Monoid Transition System (MTS). This extends the concept of

a Labelled Transition System developed by Hennessey, Olderog and others. Following Matthew Hennessy's approach we define various kinds of possible testing semantics and explore the relations between the various notions. One version of the testing semantics associates to each element of an MTS a function on the set of monoid elements. This function is the set of failures of the process.

We then consider the set of failures, show that in a natural way it is an ultrametric space, and prove its completeness. The relevance of this result is that it mathematically justifies recursion as a means of process definition. Indeed, if f is a contractive functional on the space of failures, then there is a unique p such that $p = f(p)$.

All of the mathematics that is used implicitly or explicitly in this paper has been developed using IMPS, and, in particular, all the proofs have been machine-checked. The proofs themselves are represented as runnable scripts, but for lack of space in the paper they are not included here, but are publically available along with the IMPS distribution. Some of the preliminary mathematical development is also omitted in this presentation. This includes the theory of metric-space topology up to the Contractive Fixed Point Theorem. For a discussion of this see [5].

This paper does assume a basic knowledge of the IMPS system as described in the manual [7]. In particular, the reader should be forwarned that terms may be undefined and functions may be partial, that is defined on only part of their syntactic domain.

2 Preliminaries

We begin with the definition of the theory of a single monoid. A monoid is a set with a single binary, associative operation, and an identity. Notice that in our formalization of a monoid as an IMPS theory, we are also throwing in the real numbers. This gives us the integers for free and allows us to define additional concepts which require the integers, such as sequential products.

Language 2.1 (monoid-language)
Embedded language: *h-o-real-arithmetic*
Base type: $\mathbf{U}$
Constants: $e : \mathbf{U}$
$** : [\mathbf{U} \times \mathbf{U} \rightharpoonup \mathbf{U}]$

Component theory: h-o-real-arithmetic
Top level axioms:
associative-law-for-multiplication-for-monoids
$\forall z, y, x : \mathbf{U} \quad x \cdot y \cdot z = x \cdot y \cdot z.$

right-multiplicative-identity-for-monoids $\forall x : \mathbf{U} \quad x \cdot e = x.$

left-multiplicative-identity-for-monoids $\forall x : \mathbf{U} \quad e \cdot x = x.$

anonymous total$(**, [\mathbf{U} \times \mathbf{U} \rightharpoonup \mathbf{U}])$.

Figure 1: Components and axioms for monoid-theory

Theory 2.2 (monoid-theory)
Language: *monoid-language*
Component Theories and Axioms: *See Figure 1.*

Definition 2.3 (prefix)
Theory: monoid-theory
$[\, m, n : \mathbf{U} \mapsto \exists p : \mathbf{U} \quad n = m \cdot p \,]$.

Notice that if the monoid is a group, the prefix relation is completely trivial, that is, every element is a prefix of every other element. This is in sharp contrast to the monoid of finite sequences over an alphabet in which the prefix relation is antisymmetric.

Theorem 2.4 (prefix-is-transitive)
Theory: monoid-theory
$\forall x, y, z : \mathbf{U} \quad s.\ t. \quad \text{prefix}(x, y) \wedge \text{prefix}(y, z),$
$\text{prefix}(x, z)$.

Theorem 2.5 (prefix-is-reflexive)
Theory: monoid-theory
$\forall x : \mathbf{U} \quad \text{prefix}(x, x)$.

In the case of the monoid of traces, the prefix relation which we abstractly defined corresponds exactly to the usual prefix relation for sequences. Now we define an equivalence relation between monoid elements. Monoid elements x, y satisfy this relation if either both are equal to the identity element or they have a common initial segment.

Component theory: monoid-theory
Top level axioms:
directed-property $\forall x, y, z : \mathbf{U}$ implication
- conjunction
  - $\neg(x = e)$
  - $\neg(z = e)$
  - prefix(x, y)
  - prefix(z, y)
- $\exists u : \mathbf{U}$ conjunction
  - $\neg(u = e)$
  - prefix(u, x)
  - prefix(u, z).

no-units $\forall x : \mathbf{U}$ s. t. prefix(x, e),
$x = e$.

Figure 2: Components and axioms for directed-monoid-theory

Definition 2.6 (init_eqv)
Theory: monoid-theory
$[\, a, b : \mathbf{U} \mapsto$
disjunction
- $a = e \wedge b = e$
- $\neg(a = e) \wedge \neg(b = e) \wedge (\exists x : \mathbf{U} \quad \neg(x = e) \wedge \text{prefix}(x, a) \wedge \text{prefix}(x, b))\,]$.

Theorem 2.7 (directed-monoid-theory)
Language: *the-null-language*
Component Theories and Axioms: *See Figure 2.*

Theorem 2.8 (init_eqv-reflexive-property)
Theory: monoid-theory
$\forall x : \mathbf{U}$ init_eqv(x, x).

Theorem 2.9 (init_eqv-symmetric-property)
Theory: monoid-theory
$\forall x, y : \mathbf{U} \quad \Longleftrightarrow$
- init_eqv(x, y)
- init_eqv(y, x).

Theorem 2.10 (init_eqv-transitive-property)
Theory: directed-monoid-theory
$\forall x, y, z : \mathbf{U}$ *s. t.* $\text{init\_eqv}(x, y) \wedge \text{init\_eqv}(y, z)$,
$\text{init\_eqv}(x, z)$.

Translation 2.11 (generic-theory-to-directed-monoid)
Source Theory: *generic-theory-1*
Target Theory: *directed-monoid-theory*
Sort Pairs: $\text{ind}_1 \mapsto \mathbf{U}$
This translation is a **theory interpretation**.

We now define a mapping denoted "germ," which maps the sort **U** into equivalence classes of elements of **U**. Thus for $m \in \mathbf{U}$, $\text{germ}(u)$ is a set of monoid elements. The sort **action** is then defined to be the range of the germ function. In order for these definitions to make sense, a few preliminary lemmas are needed.

In the following theorem we use the notation $t\downarrow$ to mean that the term t is defined.

Theorem 2.12 (Anonymous)
Theory: directed-monoid-theory
$\text{quotient}(\text{init\_eqv})\downarrow$.

Definition 2.13 (germ)
Theory: directed-monoid-theory
$\text{quotient}(\text{init\_eqv})$.

Theorem 2.14 (totality-of-germ)
Theory: directed-monoid-theory
$\text{total}(\text{germ}, [\mathbf{U} \rightharpoonup \text{sets}[\mathbf{U}]])$.

Theorem 2.15 (germ-equality-condition)
Theory: directed-monoid-theory
$\forall m, n : \mathbf{U} \quad \Longleftrightarrow$

- $\text{germ}(m) = \text{germ}(n)$
- disjunction
  - $m = e \wedge n = e$
  - $\exists u : \mathbf{U} \quad \neg(u = e) \wedge \text{prefix}(u, m) \wedge \text{prefix}(u, n)$.

Theorem 2.16 (Anonymous)
Theory: directed-monoid-theory
nonempty $\{ran\{\text{germ}\}\}$.

Having established that the germ function has a non empty range, we can now legitimately define the sort **action**. In IMPS sorts are defined by the predicate which recognizes which elements are elements of that sort. In this case the predicate is membership in the range of the germ function. Note that in the conventional trace framework, atomic actions are also traces whereas, in our approach, there is generally no natural inclusion mapping from actions into traces.

Sort Definition 2.17 (action)
Theory: directed-monoid-theory
$[\, s : \text{sets}[\mathbf{U}] \mapsto s \in ran\{\text{germ}\}\,]$.

3 Monoid Transition Systems

In this section, we construct a model of a set of processes. In traditional formulations of process theory, there is given a basic set of atomic actions (see [10]) which are the building blocks of processes. This set of actions is usually called the *alphabet* of the process. Let us call this set **action**. Moreover, processes p, q can be related to each other via an observable event a drawn from the set **action**. We write this ternary relation as follows $p \stackrel{a}{\Rightarrow} q$. This is thought of as the process p becoming the process q via an observable event a. This kind of structure is referred to as a *labelled transition system* [8, 9]. It is straighforward (see [8]) to define a derived relation $\rightarrow$ for triples (p, μ, q) where p, q are processes and μ is a sequence of actions.

In the model we construct in this paper, processes are related to each other by *event traces*; the event traces themselves are modeled by a monoid. This extends the concept of a labelled transition system if we regard the set of event sequences as being a monoid under concatenation.

For other different approaches to process algebra, see [3, 4, 11].

Language 3.1 (language-for-monoid-transition-system)
Embedded language: *monoid-theory*
Base type: state
Constant: act *:* $[\text{state} \times \mathbf{U} \times \text{state} \rightharpoonup *]$

Component theory: directed-monoid-theory
Top level axioms:
monoid-operation-behaves-as-composition $\forall s, t : \mathbf{U}, p, r :$
state $\iff$
- $p \xrightarrow{s \cdot t} r$
- $\exists q :$ state conjunction
  - $p \xrightarrow{s} q$
  - $q \xrightarrow{t} r$.

monoid-identity-behaves-as-identity $\forall p :$ state $p \xrightarrow{e} p$.

Figure 3: Components and axioms for monoid-transition-system

Theory 3.2 (monoid-transition-system)
Language: *language-for-monoid-transition-system*
Component Theories and Axioms: *See Figure 3.*

We now define acceptance and refusal sets for both transitions and actions.

Definition 3.3 (accepted_transitions)
Theory: monoid-transition-system
$[\, p : \text{state} \mapsto \{m : \mathbf{U} \mid \exists q : \text{state} \quad p \xrightarrow{m} q\}\,]$.

Definition 3.4 (refused_transitions)
Theory: monoid-transition-system
$[\, p : \text{state} \mapsto \{m : \mathbf{U} \mid \forall q : \text{state} \quad \neg(p \xrightarrow{m} q)\}\,]$.

Theorem 3.5 (accepted_transitions-is-prefix-closed)
Theory: monoid-transition-system
$\forall p :$ state, $m, n : \mathbf{U}$ implication
- conjunction
  - $m \in$ accepted_transitions(p)
  - prefix(n, m)
- $n \in$ accepted_transitions(p).

Definition 3.6 (accepted_actions)
Theory: monoid-transition-system

$[\, p : \text{state} \mapsto$
$\quad \{a : \text{action} \mid \exists m : \mathbf{U} \quad \text{conjunction}$
$\quad\quad \bullet\ \text{germ}(m) = a$
$\quad\quad \bullet\ m \in \text{accepted\_transitions}(p)\}\,].$

Definition 3.7 (refused_actions)
Theory: monoid-transition-system
$[\, p : \text{state} \mapsto$
$\quad \{a : \text{action} \mid \forall m : \mathbf{U} \quad s.\ t. \quad \text{germ}(m) = a,$
$\neg(m \in \text{accepted\_transitions}(p))\}\,].$

Theorem 3.8 (refused_actions-is-total)
Theory: monoid-transition-system
$\text{total}(\text{refused\_actions}, [\text{state} \rightharpoonup \text{sets}[\text{action}]]).$

Lemma 3.9 (accepted_actions-is-complement-of-refused_actions)
Theory: monoid-transition-system
$\forall p : \text{state}, x : \text{action} \quad \overline{\text{accepted\_actions}}(p) = \text{refused\_actions}(p).$

3.1 Testing Preorders

In this section we define two partial orderings on the set of all processes. We do this only to relate our approach to the existing theories of acceptance and failure semantics, even though we will not use these concepts explicitly. Indeed, of more interest to us is the concept of *testing equivalence* which we discuss in the next subsection.

Definition 3.10 (may_refuse_after)
Theory: monoid-transition-system
$[\, x : \text{sets}[\text{action}], u : \mathbf{U}, p : \text{state} \mapsto$
$\quad \exists q : \text{state} \quad \text{conjunction}$
$\quad\quad \bullet\ p \xrightarrow{u} q$
$\quad\quad \bullet\ x \subseteq \text{refused\_actions}(q)\,].$

Definition 3.11 (<_may_refuse)
Theory: monoid-transition-system
$[\, p, q : \text{state} \mapsto$
$\quad \forall x : \text{sets}[\text{action}], u : \mathbf{U} \quad s.\ t. \quad \text{may\_refuse\_after}(x, u, p),$
$\text{may\_refuse\_after}(x, u, q)\,].$

The following is one of the key concepts in the testing semantics we are investigating. The significance of this concept will be elucidated below in a series of theorems.

Definition 3.12 (failures)
Theory: monoid-transition-system
$[\, p : \text{state} \mapsto$
$\quad [\, u : \mathbf{U} \mapsto$
$\quad\quad \{s : \text{sets[action]} \mid \exists q : \text{state} \quad p \xrightarrow{u} q \wedge s \subseteq \text{refused\_actions}(q)\}\,]\,].$

Theorem 3.13 (failures-characterization-of-<_may_refuse)
Theory: monoid-transition-system
$\forall p, q : \text{state} \quad \Longleftrightarrow$
- $<_{\text{may\_refuse}}\ (p, q)$
- $\forall u : \mathbf{U} \quad \text{failures}(p)(u) \subseteq \text{failures}(q)(u).$

Definition 3.14 (must_refuse_after)
Theory: monoid-transition-system
$[\, x : \text{sets[action]}, u : \mathbf{U}, p : \text{state} \mapsto$
$\quad \forall q : \text{state} \quad s.\ t. \quad p \xrightarrow{u} q,$
$\quad x \subseteq \text{refused\_actions}(q)\,].$

Lemma 3.15 (must_refuse_after-characterization-lemma)
Theory: monoid-transition-system
$\forall x : \text{sets[action]}, u : \mathbf{U}, p : \text{state} \quad \Longleftrightarrow$
- must_refuse_after(x, u, p)
- $x \subseteq \bigcap\{[\, q : \text{state} \mapsto$
  $\quad \textit{if } p \xrightarrow{u} q \textit{ then } \text{refused\_actions}(q) \textit{ else } \text{sort\_indicator(action)}\,]\}.$

Definition 3.16 (successors_after)
Theory: monoid-transition-system
$[\, p : \text{state} \mapsto$
$\quad [\, u : \mathbf{U} \mapsto$
$\quad\quad \{a : \text{action} \mid \exists q : \text{state} \quad p \xrightarrow{u} q \wedge a \in \text{accepted\_actions}(q)\}\,]\,].$

Theorem 3.17 (successors_after-complement)
Theory: monoid-transition-system
$\forall p : \text{state}, u : \mathbf{U} \quad \bigcap\{[\, q : \text{state} \mapsto$
$\quad \textit{conditionally, if } p \xrightarrow{u} q$
- *then* refused_actions(q)
- *else* sort_indicator(action)$\,]\} = \overline{\text{successors\_after}(p)(u)}.$

Theorem 3.18 (must_refuse_after-characterization)
Theory: monoid-transition-system
$\forall x : \text{sets}[\text{action}], u : \mathbf{U}, p : \text{state} \quad \Longleftrightarrow$
- must_refuse_after(x, u, p)
- $x \subseteq \overline{(}\text{successors\_after}(p)(u)$.

Definition 3.19 (<_must_refuse)
Theory: monoid-transition-system
$[\, p, q : \text{state} \mapsto$
$\quad \forall x : \text{sets}[\text{action}], u : \mathbf{U} \quad s.\ t. \quad \text{must\_refuse\_after}(x, u, p),$
$\text{must\_refuse\_after}(x, u, q)\,]$.

Theorem 3.20 (characterization-of-<_must_refuse)
Theory: monoid-transition-system
$\forall p, q : \text{state} \quad \Longleftrightarrow$
- $<_{\text{must\_refuse}}\ (q, p)$
- $\forall u : \mathbf{U} \quad \text{successors\_after}(p)(u) \subseteq \text{successors\_after}(q)(u)$.

Definition 3.21 (must_accept_after)
Theory: monoid-transition-system
$[\, x : \text{sets}[\text{action}], u : \mathbf{U}, p : \text{state} \mapsto$
$\quad \forall q : \text{state} \quad s.\ t. \quad p \xrightarrow{u} q,$
$\neg(x \,\acute{o}\, \text{accepted\_actions}(q))\,]$.

Definition 3.22 (may_accept_after)
Theory: monoid-transition-system
$[\, x : \text{sets}[\text{action}], u : \mathbf{U}, p : \text{state} \mapsto$
$\quad \exists q : \text{state} \quad \text{conjunction}$
- $p \xrightarrow{u} q$
- $\neg(x \,\acute{o}\, \text{accepted\_actions}(q))\,]$.

Theorem 3.23 (must_accept_after-negation-of-may_refuse_after)
Theory: monoid-transition-system
$\forall p : \text{state}, x : \text{sets}[\text{action}], u : \mathbf{U} \quad \Longleftrightarrow$
- must_accept_after(x, u, p)
- $\neg(\text{may\_refuse\_after}(x, u, p))$.

Theorem 3.24 (testing-characterization-of-<_may_refuse)
Theory: monoid-transition-system

$\forall p, q$: state $\iff$
- $<_{\text{may\_refuse}}(p, q)$
- $\forall x$: sets[action], u : **U** *s. t.* must_accept_after(x, u, q), must_accept_after(x, u, p).

Theorem 3.25 (may_accept_after-negation-of-must_refuse_after)
Theory: monoid-transition-system
$\forall p$: state, x : sets[action], u : **U** $\iff$
- may_accept_after(x, u, p)
- $\neg$(must_refuse_after(x, u, p)).

Theorem 3.26 (testing-characterization-of-<_must_refuse)
Theory: monoid-transition-system
$\forall p, q$: state $\iff$
- $<_{\text{must\_refuse}}(p, q)$
- $\forall x$: sets[action], u : **U** *s. t.* may_accept_after(x, u, q), may_accept_after(x, u, p).

Theorem 3.27 (accepted_transitions-characterization-of-<_must_refuse)

Theory: monoid-transition-system
$\forall p, q$: state $\iff$
- $<_{\text{must\_refuse}}(q, p)$
- accepted_transitions$(p) \subseteq$ accepted_transitions(q).

3.2 Testing Equivalence

To complete this circle of ideas, we introduce a relation on the sort **state** and prove that the mapping $p \mapsto failures(p)$ separates precisely those pairs of points which are inequivalent.

Definition 3.28 (=_may_refuse)
Theory: monoid-transition-system
$[\, p, q$: state $\mapsto$
$\forall x$: sets[action], u : **U** $\iff$
- may_refuse_after(x, u, p)
- may_refuse_after$(x, u, q)\,]$.

Theorem 3.29 (failures-characterization-of-=_may_refuse)
Theory: graded-monoid-transition-system

$\forall p, q$: state $\iff$
- $=_{\text{may\_refuse}}(p, q)$
- failures(p) = failures(q).

3.3 Failures as Independent Objects

We would like to characterize the image of the mapping which associates to a process its possible failures. Unfortunately, this does not appear to be possible. However, we may restrict the possibilities to some extent. We begin with a useful definition.

Definition 3.30 (support)
Theory: directed-monoid-theory
$[\, f : \mathbf{U} \rightharpoonup \text{sets[sets[action]]} \mapsto \{u : \mathbf{U} \mid \text{ nonempty } \{f(u)\}\}\,]$.

The formal definition of failure which follows comprises six conditions which are suggested by the definition of the classic paper [1]. We will say u is a *trace* of the process f, if $u \in \text{support} f$. A process f may *refuse* a set of actions y after a trace u, if $y \in f(u)$.

Informally, a function

$$f : \mathbf{U} \rightharpoonup \text{sets[sets[action]]}$$

is a failure if

1. Whenever a process can refuse a set of actions y after a trace u, then it can refuse any subset of y.
2. e is always a trace.
3. Any prefix of a trace is a trace.
4. If u is a trace and a an action, then either
   - The trace u has an extension whose germ is a
   - a can always be added to any refusal set after u.
5. The function is defined for all transitions.
6. The germ of e is never refused.

Here then is the formal IMPS definition of failures:

Definition 3.31 (failure_q)
Theory: directed-monoid-theory
$[\, f : \mathbf{U} \rightharpoonup \text{sets}[\text{sets}[\text{action}]] \mapsto$
conjunction
- $\forall u : \mathbf{U}, x, y : \text{sets}[\text{action}] \quad (y \in f(u) \wedge x \subseteq y) \supset x \in f(u)$
- $e \in \text{support}(f)$
- $\forall u, v : \mathbf{U} \quad (u \in \text{support}(f) \wedge \text{prefix}(v, u)) \supset v \in \text{support}(f)$
- $\forall u : \mathbf{U}, a : \text{action} \quad u \in \text{support}(f) \supset (\exists m : \mathbf{U} \quad \text{germ}(m) = a \wedge u \cdot m \in \text{support}(f) \vee \forall x : \text{sets}[\text{action}] \quad x \in f(u) \supset x \cup \{a\} \in f(u))$
- $\text{total}(f, [\mathbf{U} \rightharpoonup \text{sets}[\text{sets}[\text{action}]]])$
- $\forall u : \mathbf{U}, s : \text{sets}[\text{action}] \quad s \in f(u) \supset \neg(\text{germ}(e) \in s)\,]$.

Lemma 3.32 (e-is-never-refused)
Theory: monoid-transition-system
$\forall p : \text{state} \quad \neg(\text{germ}(e) \in \text{refused\_actions}(p))$.

The following theorem is not a characterization of the failures of a set of processes. However, it does say that our abstract definition of failures does include all those that arise as failures of a monoid transition system.

Theorem 3.33 (range-of-failures)
Theory: monoid-transition-system
$\forall p : \text{state} \quad \text{failure}_q(\text{failures}(p))$.

Lemma 3.34 (not-everything-is-a-failure)
Theory: directed-monoid-theory
$\neg(\text{failure}_q([\, x : \mathbf{U} \mapsto \emptyset\,]))$.

Definition 3.35 (stop_ff)
Theory: directed-monoid-theory
$[\, x : \mathbf{U} \mapsto$
conditionally, if $x = e$
- *then* $\{s : \text{sets}[\text{action}] \mid \neg(\text{germ}(e) \in s)\}$
- *else* $\emptyset\,]$.

Theorem 3.36 (stop-is-a-failure)
Theory: directed-monoid-theory
$\text{failure}_q(\text{stop}_{\mathbf{F}})$.

Component theory: directed-monoid-theory
Top level axioms:
divisibility $\forall a : \mathbf{U}$ s. t. $\neg(a = e)$,
$\exists b : \mathbf{U}$ conjunction
- $\neg(b = e)$
- $\text{prefix}(b, a)$
- $0 \leq \text{lngth}(b)$
- $\text{lngth}(b) \leq 1$.

length-non-negative $\forall a : \mathbf{U} \quad 0 \leq \text{lngth}(a)$.

length-of-product $\forall a, b : \mathbf{U} \quad \text{lngth}(a \cdot b) = \text{lngth}(a) + \text{lngth}(b)$.

Figure 4: Components and axioms for graded-monoid

It is convenient to have an atomic sort denoting the set of failures. This is possible by the preceding theorem which states that the set of failures is nonempty. We will denote this sort by **F**.

Sort Definition 3.37 (F)
Theory: directed-monoid-theory
failure_q.

3.4 Graded Monoids

A graded monoid is a directed monoid with a length function "lngth." The "e" is deliberately omitted to distinguish the operator from the length function for arbitrary finite sequences. There is an IMPS-specific technical point we are slurring over here, since "length" is really a quasi-constructor, a kind of abbreviation.

Language 3.38 (language-for-graded-monoid)
Embedded language: *directed-monoid-theory*
Constant: lngth *:* $[\mathbf{U} \rightharpoonup \mathbf{R}]$

Component theories: graded-monoid monoid-transition-system **Top level axioms:** There are none.

Figure 5: Components and axioms for graded-monoid-transition-system

Theory 3.39 (graded-monoid)
Language: *language-for-graded-monoid*
Component Theories and Axioms: *See Figure 4.*

Theorem 3.40 (Anonymous-10)
Theory: graded-monoid
total(lngth, [**U** $\rightharpoonup$ **R**]).

Theorem 3.41 (lngth-of-e)
Theory: graded-monoid
lngth(e) = 0.

Theorem 3.42 (action-representatives-can-have-norm-le-1)
Theory: graded-monoid
$\forall a$: action $\exists u$: **U** conjunction
- germ(u) = a
- $0 \leq$ lngth(u)
- lngth(u) ≤ 1.

Theory 3.43 (graded-monoid-transition-system)
Language: *the-null-language*
Component Theories and Axioms: *See Figure 5.*

Definition 3.44 (eqv_may_refuse)
Theory: graded-monoid-transition-system
[p, q : state, n : **Z** $\mapsto$
$\forall x$: sets[action], u : **U** *s. t.* lngth(u) $\leq n$,
$\Longleftrightarrow$
- may_refuse_after(x, u, p)
- may_refuse_after(x, u, q)].

Theorem 3.45 (failures-characterization-of-eqv_may_refuse)
Theory: graded-monoid-transition-system
$\forall p, q : \text{state}, n : \mathbf{Z} \quad \Longleftrightarrow$
- eqv_may_refuse(p, q, n)
- $\forall u : \mathbf{U} \quad s.\ t. \quad \text{lngth}(u) \leq n,$

failures$(p)(u) =$ failures$(q)(u)$.

4 The Metric Space of Failures

In the previous section, we have associated to each element p of a monoid transition system the function $u \mapsto failures(q)(u)$ defined on the transition set, that is, the carrier set of the monoid. Moreover, this function is faithful, in that distinguishes any two processes that are observably distinct. In this section, we study more closely the space of failures and show that it has the structure of a complete metric space. In order to do this, we need some results about a class of metric-spaces we refer to as tree-like spaces. We postpone discussion of these spaces to the appendices.

There are various other approaches to constructing metric-space models for processes. One approach constructs these models as a fixed point (in a categorical sense) of a certain functor in the category of metric spaces see [2].

We begin with a completely straightfoward technical lemma which states the obvious fact that the composed function floor $\circ$ lngth maps $\mathbf{U}$ into $\mathbf{N}$.

Theorem 4.1 (Anonymous-11)
Theory: graded-monoid
floor $\circ$ lngth $\downarrow [\mathbf{U} \rightharpoonup \mathbf{N}]$.

We now apply the general theory of functions on a graded set. This theory is developed in the appendices. In the application, the underlying set is just the carrier set of the monoid, and the grading function is the floor of the length.

Translation 4.2 (functions-on-a-graded-set-to-graded-monoid)
Source Theory: *functions-on-a-graded-set*
Target Theory: *graded-monoid*
Sort Pairs: values $\mapsto$ sets[sets[action]]
Constant Pairs: degree $\mapsto$ floor $\circ$ lngth
This translation is a **theory interpretation**.

Theorem 4.3 (Anonymous-12)
Theory: graded-monoid
$\exists x : \mathbf{U} \rightharpoonup \text{sets}[\text{sets}[\text{action}]] \quad \text{total}(x, [\mathbf{U} \rightharpoonup \text{sets}[\text{sets}[\text{action}]]])$.

```
(def-theorem graded-monoid-fn%dist-triangle-inequality
  fn%dist-triangle-inequality
  (theory graded-monoid)
  (translation functions-on-a-graded-set-to-graded-monoid)
  (proof existing-theorem))

(def-theorem graded-monoid-fn%dist-symmetric
  fn%dist-symmetric
  (theory graded-monoid)
  (translation functions-on-a-graded-set-to-graded-monoid)
  (proof existing-theorem))

(def-theorem graded-monoid-fn%dist-non-negative
  fn%dist-non-negative
  (theory graded-monoid)
  (translation functions-on-a-graded-set-to-graded-monoid)
  (proof existing-theorem))

(def-theorem  graded-monoid-fn%dist-reflexive
  fn%dist-reflexive
  (theory graded-monoid)
  (translation functions-on-a-graded-set-to-graded-monoid)
  (proof existing-theorem))

(def-theory-ensemble-instances metric-spaces
  force-under-quick-load
  (target-theories graded-monoid graded-monoid)
  (permutations  (0) (0 1))
  (theory-interpretation-check using-simplification)
  (sorts (pp total%fns total%fns))
  (constants (dist fn%dist fn%dist)))
```

Theorem 4.4 (all-failures-are-total)
Theory: graded-monoid
$\forall x : \mathbf{F} \quad x \downarrow \text{total\_fns}$.

Next we view the set of failures as a subset of the metric space of total functions on **U**. In IMPS terminology, we build an interpretation from the little theory of a subspace of a metric space into the theory of a graded monoid as follows:

Translation 4.5 (ms-subspace-to-graded-monoid)
Source Theory: *ms-subspace*
Target Theory: *graded-monoid*
Sort Pairs: **P** $\mapsto$ total_fns aa $\mapsto$ **F**
Constant Pairs: dist $\mapsto$ fn_dist
This translation is a **theory interpretation**.

```
(def-theory-ensemble-instances
  metric-spaces
  force-under-quick-load
  (target-theories graded-monoid graded-monoid)
  (multiples 1 2)
  (theory-interpretation-check using-simplification)
  (sorts (pp ff ff))
  (constants (dist "lambda(x,y:ff, fn%dist(x,y))"
                   "lambda(x,y:ff, fn%dist(x,y))"))
  (special-renamings
   (complete sub%complete)
   (cauchy sub%cauchy)
   (lim sub%lim)))

(def-theorem graded-monoid-fn%approx-separation
  fn%approx-separation
  (theory graded-monoid)
  (translation functions-on-a-graded-set-to-graded-monoid)
  (proof existing-theorem))

(def-theorem graded-monoid-fn%approx-monotonicity
  fn%approx-monotonicity
  (theory graded-monoid)
  (translation functions-on-a-graded-set-to-graded-monoid)
  (proof existing-theorem))

(def-theorem graded-monoid-fn%approx-existence
  fn%approx-existence
```

```
  (theory graded-monoid)
  (translation functions-on-a-graded-set-to-graded-monoid)
  (proof existing-theorem))

(def-theorem graded-monoid-fn%approx-reflexivity
  fn%approx-reflexivity
  (theory graded-monoid)
  (translation functions-on-a-graded-set-to-graded-monoid)
  (proof existing-theorem))

(def-theorem graded-monoid-fn%approx-symmetry
  fn%approx-symmetry
  (theory graded-monoid)
  (translation functions-on-a-graded-set-to-graded-monoid)
  (proof existing-theorem))

(def-theorem graded-monoid-fn%approx-transitivity
  fn%approx-transitivity
  (theory graded-monoid)
  (translation functions-on-a-graded-set-to-graded-monoid)
  (proof existing-theorem))
```

Translation 4.6 (degree-equivalence-to-graded-monoid)
Source Theory: *degree-equivalence*
Target Theory: *graded-monoid*
Sort Pairs: $\mathbf{P} \mapsto$ total_fns
Constant Pairs: approx $\mapsto$ fn_approx
This translation is a **theory interpretation**.

Theorem 4.7 (characterization-ultrametric-limit-of-fns)
Theory: graded-monoid
$\forall f : \mathbf{Z} \rightharpoonup \text{total\_fns}, s : \text{total\_fns} \quad \Longleftrightarrow$
- $\lim f = s$
- $\forall m : \mathbf{Z} \quad s.\ t. \quad 0 \leq m,$

$\exists n : \mathbf{Z} \quad \forall p : \mathbf{Z}, u : \mathbf{U} \quad s.\ t. \quad n \leq p \wedge \text{lngth}(u) < m + 1,$
$f(p)(u) = s(u).$

Lemma 4.8 (prefix-has-smaller-length-lemma)
Theory: graded-monoid
$\forall a, b : \mathbf{U} \quad s.\ t. \quad \text{prefix}(a, b),$
$\text{lngth}(a) \leq \text{lngth}(b).$

Theorem 4.9 (failure_q-is-closed-under-lim)
Theory: graded-monoid
$\forall s : \mathbf{Z} \rightharpoonup$ total_fns implication
- conjunction
  - $\lim s \downarrow$
  - $\forall n : \mathbf{Z} \quad s(n) \downarrow \supset \text{failure}_q(s(n))$
- $\text{failure}_q(\lim s)$.

The following is the basic result which allows recursive definitions of processes. Recall that a metric space is *complete* if and only every Cauchy sequence in it converges. In the following theorem, the constant sub_complete is an element of sort **prop**. Its meaning is that the set of elements in **F**, in the metric induced from the space of all functions

$$\mathbf{U} \rightharpoonup \text{sets[sets[action]]}$$

is complete. Notice that in informal parlance one would have just said **F** is complete, but such distinctions are one of the burdens of formal mathematics.

Theorem 4.10 (completeness-of-ff)
Theory: graded-monoid
sub_complete.

Component theory: h-o-real-arithmetic
Top level axioms:
approx-separation $\forall x, y : \mathbf{P} \quad$ s. t. $\quad \neg(x = y)$,
$\exists n : \mathbf{Z} \quad \neg(\text{approx}(x, y, n))$.

approx-monotonicity $\forall m, n : \mathbf{Z}, x, y : \mathbf{P} \quad$ s. t. $\quad \text{approx}(x, y, n) \wedge$
$m \leq n$,
$\text{approx}(x, y, m)$.

approx-existence $\forall x, y : \mathbf{P} \quad \exists m : \mathbf{Z} \quad \text{approx}(x, y, m)$.

approx-reflexivity $\forall m : \mathbf{Z}, x : \mathbf{P} \quad \text{approx}(x, x, m)$.

approx-symmetry $\forall m : \mathbf{Z}, x, y : \mathbf{P} \quad$ s. t. $\quad \text{approx}(x, y, m)$,
$\text{approx}(y, x, m)$.

approx-transitivity $\forall m : \mathbf{Z}, x, y, z : \mathbf{P} \quad$ s. t. $\quad \text{approx}(x, y, m) \wedge$
$\text{approx}(y, z, m)$,
$\text{approx}(x, z, m)$.

Figure 6: Components and axioms for degree-equivalence

A Tree-like Metric Spaces

In this section we discuss sets $\mathbf{P}$ which are tree-like in the following sense: for each integer n, there is a relation of *equality of degree* n on the set $\mathbf{P}$. Of course, we assume this relation satisfies some natural equivalence properties. The set of paths through a labelled tree form a natural example of such a structure. In this case, we say paths p, p' are equal of order n if they coincide up to the step n.

In this section we assume basic facts about metric spaces.

Language A.1 (degree-equivalence-language)
Embedded language: *h-o-real-arithmetic*
Base type: $\mathbf{P}$
Constant: approx *:* $[\mathbf{P} \times \mathbf{P} \times \mathbf{Z} \rightharpoonup *]$

Theory A.2 (degree-equivalence)

Language: *degree-equivalence-language*
Component Theories and Axioms: *See Figure 6.*

Definition A.3 (sep%deg)
Theory: degree-equivalence
$[\, x, y : \mathbf{P} \mapsto$
$\quad \iota\, n : \mathbf{Z} \quad$ conjunction
$\quad\quad \bullet\ \neg(\text{approx}(x, y, n))$
$\quad\quad \bullet\ \forall m : \mathbf{Z} \quad m < n \supset \text{approx}(x, y, m)\,]$.

Theorem A.4 (iota-free-characterization-of-sep%deg)
Theory: degree-equivalence
$\forall x, y : \mathbf{P}, n : \mathbf{Z} \quad \Longleftrightarrow$
- $\text{sep\_deg}(x, y) = n$
- conjunction
  - $\neg(\text{approx}(x, y, n))$
  - $\forall m : \mathbf{Z} \quad m < n \supset \text{approx}(x, y, m)$.

Theorem A.5 (alternate-iota-free-characterization-of-sep%deg)
Theory: degree-equivalence
$\forall x, y : \mathbf{P}, n : \mathbf{Z} \quad \Longleftrightarrow$
- $\text{sep\_deg}(x, y) = n$
- conjunction
  - $\neg(\text{approx}(x, y, n))$
  - $\text{approx}(x, y, n - 1)$.

Theorem A.6 (definedness-of-sep%deg)
Theory: degree-equivalence
$\forall x, y : \mathbf{P} \quad s.\ t. \quad \neg(x = y)$,
$\text{sep\_deg}(x, y) \downarrow$.

Theorem A.7 (undefinedness-case-of-sep%deg)
Theory: degree-equivalence
$\forall x : \mathbf{P} \quad \neg(\text{sep\_deg}(x, x) \downarrow)$.

Theorem A.8 (sep%deg-upper-bound)
Theory: degree-equivalence
$\forall x, y : \mathbf{P}, n : \mathbf{Z} \quad s.\ t. \quad \neg(x = y)$,
$\Longleftrightarrow$
- $\text{sep\_deg}(x, y) \leq n$
- $\neg(\text{approx}(x, y, n))$.

Lemma A.9 (symmetry-of-sep%deg)
Theory: degree-equivalence
$\forall x, y : \mathbf{P} \quad s.\ t. \quad \neg(x = y)$,
$\quad \text{sep\_deg}(x, y) = \text{sep\_deg}(y, x)$.

Lemma A.10 (reverse-ultrametric-lemma)
Theory: degree-equivalence
$\forall x, y, z : \mathbf{P}$ implication
- conjunction
  - $\neg(x = y)$
  - $\neg(y = z)$
  - $\neg(x = z)$
  - $\text{sep\_deg}(x, y) \leq \text{sep\_deg}(y, z)$
- $\text{sep\_deg}(x, y) \leq \text{sep\_deg}(x, z)$.

Lemma A.11 (reverse-ultrametric-inequality)
Theory: degree-equivalence
$\forall x, y, z : \mathbf{P} \quad s.\ t. \quad \neg(x = y) \wedge \neg(y = z) \wedge \neg(x = z)$,
$\quad \min(\text{sep\_deg}(x, y), \text{sep\_deg}(y, z)) \leq \text{sep\_deg}(x, z)$.

Definition A.12 (sep%dist)
Theory: degree-equivalence
$[\, x, y : \mathbf{P} \mapsto$
conditionally, if $x = y$
- *then* 0
- *else* $2^{-\text{sep\_deg}(x,y)}$].

Theorem A.13 (sep%dist-reflexive)
Theory: degree-equivalence
$\forall x, y : \mathbf{P} \quad \iff$
- $\text{sep\_dist}(x, y) = 0$
- $x = y$.

Theorem A.14 (sep%dist-non-negative)
Theory: degree-equivalence
$\forall x, y : \mathbf{P} \quad 0 \leq \text{sep\_dist}(x, y)$.

Theorem A.15 (sep%dist-symmetric)
Theory: degree-equivalence
$\forall x, y : \mathbf{P} \quad \text{sep\_dist}(x, y) = \text{sep\_dist}(y, x)$.

Theorem A.16 (min-under-nondecreasing-fn-lemma)
Theory: h-o-real-arithmetic
$\forall x, y : \mathbf{Z}, f : \mathbf{Z} \rightharpoonup \mathbf{R}$ implication
- $\forall x, y : \mathbf{Z}$ *s. t.* $x \leq y$,
  $f(y) \leq f(x)$
- $f(\min(x, y)) = \max(f(x), f(y))$.

Theorem A.17 (sep%dist-ultrametric)
Theory: degree-equivalence
$\forall x, y, z : \mathbf{P}$ $\text{sep\_dist}(x, z) \leq \max(\text{sep\_dist}(x, y), \text{sep\_dist}(y, z))$.

Translation A.18 (ultrametric-to-degree-equivalence)
Source Theory: *ultrametric-spaces*
Target Theory: *degree-equivalence*
Constant Pairs: dist $\mapsto$ sep_dist
This translation is a **theory interpretation**.

```
(def-theorem sep%dist-is-a-metric
  ultrametic-spaces-are-metric
  (theory degree-equivalence)
  (translation ultrametric-to-degree-equivalence)
  (proof existing-theorem))

(def-theory-ensemble-instances metric-spaces
  force-under-quick-load
  (target-theories degree-equivalence degree-equivalence)
  (permutations  (0) (0 1))
  (theory-interpretation-check using-simplification)
  (sorts (pp pp pp))
  (constants (dist sep%dist sep%dist)))
```

Lemma A.19 (small-distance-characterization-lemma)
Theory: degree-equivalence
$\forall x, y : \mathbf{P}, n : \mathbf{Z}$ $\iff$
- $\text{sep\_dist}(x, y) \leq 2^{-n}$
- disjunction
  - $x = y$
  - $n \leq \text{sep\_deg}(x, y)$.

Theorem A.20 (small-distance-characterization)
Theory: degree-equivalence
$\forall x, y : \mathbf{P}, n : \mathbf{Z} \quad \Longleftrightarrow$
- sep_dist$(x, y) \leq 2^{-n}$
- approx$(x, y, n-1)$.

Lemma A.21 (powers-arbitrarily-small)
Theory: h-o-real-arithmetic
$\forall r, \epsilon : \mathbf{R} \quad s.\ t. \quad 0 < r \wedge r < 1 \wedge 0 < \epsilon,$
$\exists n : \mathbf{Z} \quad$ conjunction
- $1 \leq n$
- $r^n \leq \epsilon$.

Theorem A.22 (cauchy-characterization-for-sep%dist)
Theory: degree-equivalence
$\forall f : \mathbf{Z} \rightharpoonup \mathbf{P} \quad \Longleftrightarrow$
- cauchy(f)
- $\forall m : \mathbf{Z} \quad \exists n : \mathbf{Z} \quad \forall p : \mathbf{Z} \quad s.\ t. \quad n \leq p,$
  approx$(f(p), f(p+1), m)$.

Theorem A.23 (strong-cauchy-characterization-for-sep%dist)
Theory: degree-equivalence
$\forall f : \mathbf{Z} \rightharpoonup \mathbf{P} \quad \Longleftrightarrow$
- cauchy(f)
- $\forall m : \mathbf{Z} \quad \exists n : \mathbf{Z} \quad \forall p, q : \mathbf{Z} \quad s.\ t. \quad n \leq p \wedge n \leq q,$
  approx$(f(p), f(q), m)$.

Theorem A.24 (lim-characterization-for-sep%dist)
Theory: degree-equivalence
$\forall f : \mathbf{Z} \rightharpoonup \mathbf{P}, s : \mathbf{P} \quad \Longleftrightarrow$
- $\lim f = s$
- $\forall m : \mathbf{Z} \quad \exists n : \mathbf{Z} \quad \forall p : \mathbf{Z} \quad s.\ t. \quad n \leq p,$
  approx$(f(p), s, m)$.

The next theorem gives a characterization of contractive mappings. It states that a mapping f is contractive if it always increases the degree of approximation between pairs of elements.

Theorem A.25 (contraction-characterization-for-sep%dist)
Theory: degree-equivalence

Component theory: h-o-real-arithmetic **Top level axioms:** **totality-of-degree** total(degree, [**U** $\rightharpoonup$ **N**]).

Figure 7: Components and axioms for functions-on-a-graded-set

$\forall f : \mathbf{P} \rightharpoonup \mathbf{P}$ implication
- $\forall x, y : \mathbf{P}, m : \mathbf{Z}$ *s. t.* approx(x, y, m), approx$(f(x), f(y), m + 1)$
- contraction(f).

B Function Spaces

In this section we apply the preceeding theory to put a metric on certain function spaces. Basically, under this metric, functions are close if they are equal for large elements.

Language B.1 (functions-on-a-graded-set)
Embedded language: *h-o-real-arithmetic*
Base types: **U** values
Constant: degree *:* [**U** $\rightharpoonup$ **N**]

Theory B.2 (functions-on-a-graded-set)
Language: *functions-on-a-graded-set*
Component Theories and Axioms: *See Figure 7.*

Theorem B.3 (Anonymous) Theory: functions-on-a-graded-set
$\exists x : \mathbf{U} \rightharpoonup$ values total$(x, [\mathbf{U} \rightharpoonup$ values$])$.

Sort Definition B.4 (total%fns)
Theory: functions-on-a-graded-set
$[\, f : \mathbf{U} \rightharpoonup$ values $\mapsto$
total$(f, [\mathbf{U} \rightharpoonup$ values$])\,]$.

Definition B.5 (fn%approx)
Theory: functions-on-a-graded-set
$[\, f, g : \text{total\_fns}, n : \mathbf{Z} \mapsto$
$\quad \forall k : \mathbf{U} \quad s.\ t. \quad \text{degree}(k) \leq n,$
$\; f(k) = g(k)\,].$

Lemma B.6 (fn%approx-separation)
Theory: functions-on-a-graded-set
$\forall x, y : \text{total\_fns} \quad s.\ t. \quad \neg(x = y),$
$\; \exists n : \mathbf{Z} \quad \neg(\text{fn\_approx}(x, y, n)).$

Lemma B.7 (fn%approx-monotonicity)
Theory: functions-on-a-graded-set
$\forall m : \mathbf{Z}, x, y : \text{total\_fns} \quad$ implication
- $\exists n : \mathbf{Z} \quad$ conjunction
  - $\text{fn\_approx}(x, y, n)$
  - $m \leq n$
- $\text{fn\_approx}(x, y, m).$

Lemma B.8 (fn%approx-existence)
Theory: functions-on-a-graded-set
$\forall x, y : \text{total\_fns} \quad \exists m : \mathbf{Z} \quad \text{fn\_approx}(x, y, m).$

Lemma B.9 (fn%approx-reflexivity)
Theory: functions-on-a-graded-set
$\forall m : \mathbf{Z}, x : \text{total\_fns} \quad \text{fn\_approx}(x, x, m).$

Lemma B.10 (fn%approx-symmetry)
Theory: functions-on-a-graded-set
$\forall m : \mathbf{Z}, x, y : \text{total\_fns} \quad s.\ t. \quad \text{fn\_approx}(x, y, m),$
$\; \text{fn\_approx}(y, x, m).$

Theorem B.11 (fn%approx-transitivity)
Theory: functions-on-a-graded-set
$\forall m : \mathbf{Z}, x, z : \text{total\_fns} \quad$ implication
- $\exists y : \text{total\_fns} \quad$ conjunction
  - $\text{fn\_approx}(x, y, m)$
  - $\text{fn\_approx}(y, z, m)$
- $\text{fn\_approx}(x, z, m).$

Translation B.12 (degree-equivalence-to-functions-on-a-graded-set)

Source Theory: *degree-equivalence*
Target Theory: *functions-on-a-graded-set*
Sort Pairs: $\mathbf{P} \mapsto$ total_fns
Constant Pairs: approx $\mapsto$ fn_approx
This translation is a **theory interpretation**.

Under this translation, some constants such as the predicate "cauchy" get translated into long-winded expressions, simply because the theory functions-on-a-graded-set has not yet been viewed as a metric space. We now do this.

```
(def-theory-ensemble-instances metric-spaces
  force-under-quick-load
  (target-theories functions-on-a-graded-set functions-on-a-graded-set)
  (permutations  (0) (0 1))
  (theory-interpretation-check using-simplification)
  (sorts (pp total%fns total%fns))
  (constants (dist fn%dist fn%dist)))
```

Definition B.13 (discrete%lim)
Theory: functions-on-a-graded-set
$[\, s : \mathbf{Z} \rightharpoonup \text{total\_fns} \mapsto$
$\quad [\, u : \mathbf{U} \mapsto$
$\qquad s(\text{set\_min}(\{n : \mathbf{Z} \mid 0 \leq n \wedge (\forall p, q : \mathbf{Z} \quad (n \leq p \wedge n \leq q) \supset$
$\qquad \text{fn\_approx}(s(p), s(q), \text{degree}(u)))\}))(u)\,]\,]$.

Theorem B.14 (definedness-of-discrete%lim)
Theory: functions-on-a-graded-set
$\forall s : \mathbf{Z} \rightharpoonup \text{total\_fns} \quad s.\ t. \quad \text{cauchy}(s)$,
$\text{discrete\_lim}(s) \downarrow \text{total\_fns}$.

Theorem B.15 (discrete%lim-eventually-constant-property)
Theory: functions-on-a-graded-set
$\forall s : \mathbf{Z} \rightharpoonup \text{total\_fns} \quad s.\ t. \quad \text{discrete\_lim}(s) \downarrow \text{total\_fns}$,
$\forall u : \mathbf{U} \quad \exists m : \mathbf{Z} \quad \forall p : \mathbf{Z} \quad s.\ t. \quad m \leq p$,
$s(p)(u) = \text{discrete\_lim}(s)(u)$.

Lemma B.16 (cauchy-implies-discrete%lim-is-lim)
Theory: functions-on-a-graded-set

$\forall s : \mathbf{Z} \rightharpoonup \text{total\_fns} \quad s.\ t. \quad \text{cauchy}(s)$,
$\lim s = \text{discrete\_lim}(s)$.

Theorem B.17 (completeness-of-total%fns)
Theory: functions-on-a-graded-set
complete.

C The μ operator

The μ operator associates to each contractive functional its unique fixed point.

Definition C.1 (mu)
Theory: metric-spaces
$[\, f : \mathbf{P} \rightharpoonup \mathbf{P} \mapsto \iota x : \mathbf{P} \quad f(x) = x \,]$.

Definition C.2 (contraction)
Theory: metric-spaces
$[\, f : \mathbf{P} \rightharpoonup \mathbf{P} \mapsto$
$\exists r : \mathbf{R}$ conjunction
- $0 < r$
- $r < 1$
- $\forall x, y : \mathbf{P} \quad (x \in dom\{f\} \wedge y \in dom\{f\}) \supset \text{dist}(f(x), f(y)) \leq r \cdot \text{dist}(x, y)$].

Theorem C.3 (iota-free-characterization-of-mu)
Theory: metric-spaces
$\forall f : \mathbf{P} \rightharpoonup \mathbf{P}, r : \mathbf{R}, x : \mathbf{P} \quad s.\ t. \quad \text{contraction}(f)$,
$\Longleftrightarrow$
- $\mu(f) = x$
- $f(x) = x$.

Theorem C.4 (definedness-of-mu-for-contractions)
Theory: metric-spaces
$\forall f : \mathbf{P} \rightharpoonup \mathbf{P}, r : \mathbf{R}, x : \mathbf{P}$ implication
- conjunction
  - complete
  - contraction(f)
  - total($f, [\mathbf{P} \rightharpoonup \mathbf{P}]$)
- $\mu(f) \downarrow$.

```
(def-theory-ensemble-instances
  metric-spaces
  force-under-quick-load
  (permutations (0))
  (sorts (pp bfun))
  (constants (dist bfun%dist))
  (target-theories mappings-into-a-pointed-metric-space)
  (special-renamings
   (ball bfun%ball)
   (complete bfun%complete)
   (lipschitz%bound%on bfun%lipschitz%bound%on)
   (lipschitz%bound bfun%lipschitz%bound)))
```

Theorem C.5 (definedness-of-mu-for-contractions-on-functions)
Theory: mappings-into-a-pointed-metric-space
$\forall f : \text{bfun} \rightharpoonup \text{bfun}$ implication
- conjunction
  - complete
  - contraction(f)
  - total(f, [bfun $\rightharpoonup$ bfun])
- $\mu(f) \downarrow$.

Theorem C.6 (condition-for-contractions-on-function-spaces)
Theory: mappings-into-a-pointed-metric-space
$\forall f : \text{bfun} \rightharpoonup \text{bfun}$ implication
- $\exists k : \mathbf{R}$ conjunction
  - $0 < k$
  - $k < 1$
  - $\forall \phi, \psi : \text{bfun}, x : \text{ind}_1 \quad \exists y : \text{ind}_1 \quad \text{dist}(f(\phi)(x), f(\psi)(x)) \leq k \cdot \text{dist}(\phi(y), \psi(y))$
- contraction(f).

References

[1] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *JACM*, 31(3):560–599, 1984.

[2] J. W. de Bakker and J. I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54:70–120, 1982.

[3] R. DeNicola. Extensional equivalences for transition systems. *Acta Informatica*, 24, 1987.

[4] R. DeNicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34, 1984.

[5] W. M. Farmer, J. D. Guttman, and F. J. Thayer. Little theories. In D. Kapur, editor, *Automated Deduction—CADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 567–581. Springer-Verlag, 1992.

[6] W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: an Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11:213–248, October 1993.

[7] W. M. Farmer, J. D. Guttman, and F. J. Thayer. The IMPS user's manual. Technical Report M93B-138, The MITRE Corporation, Bedford, MA, November 1993.

[8] M. Hennessy. Acceptance trees. *JACM*, 32(4):896–928, 1985.

[9] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[10] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, NJ, 1985.

[11] E.-R. Olderog and C. A. R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23, 1986.