Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 249 (2009) 357–375
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Exploratory Functions on Nondeterministic Strategies, up to Lower Bisimilarity ⋆

Paul Blain Levy[1] Kidane Yemane Weldemariam[2]

University of Birmingham, Birmingham, B15 2TT, UK

Abstract

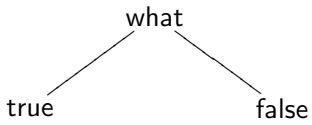
We consider a typed lambda-calculus with no function types, only alternating sum and product types, so that closed terms represent strategies. We add nondeterminism and consider strategies up to lower (i.e. divergence-insensitive) bisimilarity. We investigate the question: when is a function on strategies definable by an open term (with sufficiently large nondeterminism)?
The answer is: when it is "exploratory". This is a kind of iterated continuity property, coinductively defined, that is decidable in the case of a function between finite types.
In particular, any exploratory function between countably nondeterministic strategies is definable by a continuum nondeterministic term.

Keywords: strategy, lambda calculus, exploratory, nondeterminism, bisimilarity

1 Introduction

1.1 Functions between strategies

We consider games in which play alternates between two players, Opponent and Proponent. Such a game may be represented as a countable forest. Two examples are A, in which Opponent starts:

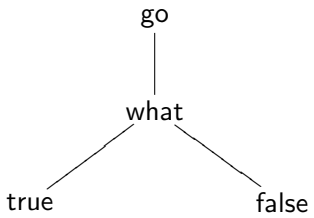


⋆ supported by EPSRC grants EP/C536827/1 and EP/E056091/1

[1] Email: pbl@cs.bham.ac.uk
[2] Email: kidane.yemane@gmail.com

doi:10.1016/j.entcs.2009.07.098

and $\underline{B}$, in which Proponent starts:



A *deterministic strategy* for a game is a rule telling Proponent how to play, no matter how Opponent plays. Proponent always has the option of simply diverging, and play may continue forever (we are not considering any notion of winning). The set of deterministic strategies for a game form a domain. For example, writing $\mathbb{B} \stackrel{\text{def}}{=} \{\mathsf{true}, \mathsf{false}\}$, the domains for A and $\underline{B}$ are $\mathbb{B}_\perp$ and $(\mathbb{B}_\perp)_\perp$ respectively.

Suppose that f is a function from deterministic strategies for one game to those for another. Under what conditions is f definable by a program? To answer this question, we need a programming language that converts strategies into strategies. To simplify our question, we shall ignore first-order computability, i.e. we assume that each function $\mathbb{N} \longrightarrow \mathbb{N}$, even if non-computable, is provided as a constant in the language.

- If the language provides parallel-or and parallel-exists operators [8], then f is definable iff it is continuous. This follows from the "universality" result of [8,9].
- If the language is purely sequential, then f is definable iff it is Kahn-Plotkin sequential [3].

Let us consider next *nondeterministic* strategies for a game. The first problem here is that it is debatable when two nondeterministic strategies should be deemed equal. For example, under may-testing equivalence, a nondeterministic strategy can be represented as a set of finite traces. In that case, a function f is definable iff it is continuous. Another possibility is infinite trace equivalence [5].

In this paper, we equate two nondeterministic strategies when they are bisimilar; more specifically, when they are *lower bisimilar*, meaning that (as in may-testing equivalence) we ignore the possibility of divergence. Under this equivalence, the set of strategies for A is $\mathcal{P}\mathbb{B}$ and the set of strategies for $\underline{B}$ is $\mathcal{P}\mathcal{P}\mathbb{B}$.

We again want to know: if f is a function mapping strategies to strategies, when is it definable by a program? For example, the following table describes a function $\mathcal{P}\mathbb{B} \xrightarrow{f} \mathcal{P}\mathcal{P}\mathbb{B}$ that maps a strategy for A to a strategy for $\underline{B}$.

$$\begin{aligned} \{\} &\mapsto \{\{\}\} \\ \{\mathsf{true}\} &\mapsto \{\{\mathsf{true}\}\} \\ \{\mathsf{false}\} &\mapsto \{\{\mathsf{false}\}\} \\ \{\mathsf{true}, \mathsf{false}\} &\mapsto \{\{\mathsf{true}\}, \{\mathsf{false}\}\} \end{aligned}$$

We are going to argue that f is not definable by a program. Suppose it is definable by a term M, with a free identifier $\mathtt{x}$ representing the argument. Each of the four argument can be represented by an appropriate term:

`diverge`	represents	$\{\}$
`true`	represents	$\{\mathsf{true}\}$
`false`	represents	$\{\mathsf{false}\}$
`true or false`	represents	$\{\mathsf{true}, \mathsf{false}\}$

Now $M[\texttt{diverge}/\texttt{x}]$ may, by making certain choices, return $\langle \mathsf{go}, W\rangle$. So for any argument P, the term $M[P/\texttt{x}]$ may make the same choices and return $\langle \mathsf{go}, W_P\rangle$. We know that $W_{\texttt{true}}$ in response to what may return `true`. So $W_{\texttt{true or false}}$ in response to what may return `true`—and, by the same argument, may return `false`. This contradicts the last line of the table.

Is there some mathematical condition that every function defined by a nondeterministic program must satisfy, that f fails? The answer cannot be a monotonicity requirement, because any set of three lines of our table *can* be realized by a program [3]. And it is unlikely to be a continuity condition, because the sets in this example are finite.

In this paper, we show that (provided the defining program may use nondeterminism of unrestricted cardinality) a function f on strategies is definable iff it is *exploratory*. This is a kind of iterated continuity condition, defined coinductively. If f applied to x plays a move and then continues as a strategy y, then that move must be obtained by exploring x to a finite degree—that much is continuity—but there must be another function g that gives y, and so forth. When the games in question are finite, as in our example, exploratoriness is a decidable property.

1.2 Outline of Paper

In Sect. 2, we define a calculus for programs that manipulate strategies; this enables us to precisely formulate the definability problem. In Sect. 3 we give a result about operational semantics, the *syntactic exploration* theorem, and convert this into a condition on functions, solving our problem. In Sect. 4, we modify our calculus to make it *affine*, in the style of [7], and adapt our results accordingly. Finally in Sect. 5 we discuss some possible future directions.

Note on cardinals We distinguish between sets and classes. If A is a class, then $\mathcal{P}(A)$ is the class of subsets and $\mathcal{P}_{\leqslant \aleph_0}(A)$ the class of subsets of size $\leqslant \aleph_0$.

Acknowledgements We thank Andreas Blass and Stefan Milius for their help with cardinality questions.

[3] Writing x for the argument, here are the four programs, listed in order of the omitted line.

(i) Apply x to what. If you get a boolean b, play go and then, if asked what, play b.

(ii) Nondeterministic choice between the following.
- Play go. Then, if asked what, play what on x. If you get true, then diverge, and if you get false, then play false.
- Play what on x. If you get true, then play go, and then, if asked what, play true. If you get false, then diverge.

(iii) The same as (ii), but with true and false exchanged.

(iv) Play go. Then, if asked what, play what on x. If you get a boolean b, play b.

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{c}} \langle \hat{\imath}, V\rangle : \sum_{i\in I} A_i}\ \hat{\imath} \in I \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \sum_{i\in I} A_i \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{c}} N_i : \underline{B}\ \ (\forall i \in I)}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ M\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle . N_i\}_{i\in I} : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M_i : \underline{B}_i\ \ (\forall i \in I)}{\Gamma \vdash^{\mathsf{v}} \lambda\{i.M_i\}_{i\in I} : \prod_{i\in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \prod_{i\in I} \underline{B}_i}{\Gamma \vdash^{\mathsf{c}} V\hat{\imath} : \underline{B}_{\hat{\imath}}}\ \hat{\imath} \in I$$

$$\frac{}{\Gamma \vdash^{\mathsf{v}} \mathtt{x} : A}\ (\mathtt{x} : A) \in \Gamma \qquad \frac{\Gamma \vdash^{\mathsf{c}} M_j : \underline{B} \quad (\forall j < \alpha)}{\Gamma \vdash^{\mathsf{c}} \mathtt{choose}_{j<\alpha} M_j : \underline{B}}\ (\alpha \text{ a cardinal} > 0)$$

Fig. 1. Syntax of intuitionistic strategy calculus, with unrestricted nondeterministm

2 Strategy Calculus—The Intuitionistic Version

2.1 Defining the Calculus

The strategy calculus is essentially a fragment of typed λ-calculus with countable sum and countable product types. This fragment does not contain function types. The sum and product constructors alternate, so there are two kinds of type: value type (representing an Opponent-first game) and computation type (representing a Proponent-first game). They are defined coinductively, so the type syntax is non-well-founded. The definition is as follows (we underline computation types):

$$\begin{aligned} \text{value type } A &::= \prod_{i\in I} \underline{B}_i \\ \text{computation type } \underline{B} &::= \sum_{i\in I} A_i \end{aligned}$$

where each set I of *tags* is countable (let us say: a subset of $\mathbb{N}$).

A *typing context* Γ is a finite set of identifiers, each given a value type. We write $\Gamma \vdash^{\mathsf{v}} V : A$ to say that V is a value, and $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ to say that M is a computation. The syntax is given in Fig. 1; this too is a coinductive definition, so the term syntax is non-well-founded. We write `pm` for pattern-match.

This calculus is *intuitionistic*, in the sense that weakening and contraction are admissible. In Sect. 4 we consider an *affine* variant, where contraction is excluded.

Because terms may include nondeterminism of arbitrary cardinality, they form a class but not a set. A term M is *countably branching* when all the nondeterminism cardinals appearing in M are $\leqslant \aleph_0$.

A *terminal computation* is one of the form $\langle \hat{\imath}, V\rangle$. We inductively define a *convergence* relation $M \Downarrow T$ where M is a closed computation and T a terminal computation of the same type. This is presented in Fig. 2. We could also define a *divergence* predicate $M \Uparrow$, but this paper ignores divergence.

Proposition 2.1 *Let $\vdash^{\mathsf{c}} M : \underline{B}$ be countably branching. Then $\{T \mid M \Downarrow T\}$ is countable.*

Proof. For each $n \in \mathbb{N}$, the set of T such that $M \Downarrow T$ has a proof of height $< n$ is countable. This is proved by induction on n. The result follows. □

$$\frac{}{\langle \hat{\imath}, V\rangle \Downarrow \langle \hat{\imath}, V\rangle} \qquad \frac{M_{\hat{\imath}} \Downarrow T}{\lambda\{i.M_i\}\hat{\imath} \Downarrow T}$$

$$\frac{M \Downarrow \langle \hat{\imath}, V\rangle \quad N_{\hat{\imath}}[V/\mathtt{x}] \Downarrow T}{\mathtt{pm}\ M\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.N_i\}_{i\in I} \Downarrow T} \qquad \frac{M_{\hat{\jmath}} \Downarrow T}{\mathtt{choose}_{j<\alpha} M_j \Downarrow T}\ \hat{\jmath} < \alpha$$

Fig. 2. Big-step semantics of strategy calculus

We write $\Gamma \vdash^{\mathsf{c}} \mathtt{diverge} : \underline{B}$ for some computation that does not converge to anything. Given a family of computations $\Gamma \vdash^{\mathsf{c}} M_i : \underline{B}$ indexed by $i \in I$, we define $\Gamma \vdash^{\mathsf{c}} \mathtt{choose}^{\mathtt{div}}\{M_i\}_{i\in I} : \underline{B}$ to be to be $\mathtt{diverge}$ if I is empty, and otherwise to be $\mathtt{choose}\ \{M_{i(j)}\}_{j<|I|}$, where i is some bijection from the cardinals $< |I|$ to I.

The closed terms of our calculus form the nodes of a labelled transition system, with transitions given by $M \xrightarrow{\hat{\imath}} V$, whenever $M \Downarrow \langle \hat{\imath}, V\rangle$, and $V \xrightarrow{\hat{\imath}} V\hat{\imath}$. We therefore use "syntactic node" as a synonym for "closed term".

Definition 2.2 For any type A, we write $\mathsf{Syn}(A)$ for the class of syntactic nodes of type A, and $\mathsf{Syn}_{\leqslant\aleph_0}(A)$ for the set of countably branching syntactic nodes. We define functions [4]

$$\begin{array}{rcl} \mathsf{Syn}(\sum_{i\in I} A_i) & \overset{\alpha(\sum_{i\in I} A_i)}{\longrightarrow} & \mathcal{P}\sum_{i\in I}\mathsf{Syn}(A_i) \\ M & \longmapsto & \{(i, V) \mid M \Downarrow \langle i, V\rangle\} \end{array}$$

$$\begin{array}{rcl} \mathsf{Syn}(\prod_{i\in I} \underline{B}_i) & \overset{\alpha(\prod_{i\in I} \underline{B}_i)}{\longrightarrow} & \prod_{i\in I}\mathsf{Syn}(\underline{B}_i) \\ V & \longmapsto & i \mapsto Vi \end{array}$$

These restrict to functions

$$\mathsf{Syn}_{\leqslant\aleph_0}(\textstyle\sum_{i\in I} A_i) \overset{\alpha_{\leqslant\aleph_0}(\sum_{i\in I} A_i)}{\longrightarrow} \mathcal{P}_{\leqslant\aleph_0}\sum_{i\in I}\mathsf{Syn}_{\leqslant\aleph_0}(A_i)$$

$$\mathsf{Syn}_{\leqslant\aleph_0}(\textstyle\prod_{i\in I} \underline{B}_i) \overset{\alpha_{\leqslant\aleph_0}(\prod_{i\in I} \underline{B}_i)}{\longrightarrow} \prod_{i\in I}\mathsf{Syn}_{\leqslant\aleph_0}(\underline{B}_i)$$

2.2 Bisimilarity

We have seen that the syntactic nodes form a transition system. This gives a notion of similarity and bisimilarity, which we now describe in detail.

Definition 2.3 Let $\mathcal{R}$ be a *type-indexed relation*, i.e. a binary relation between the closed terms of each type. It is a *lower simulation* when

- $V\ \mathcal{R}\ V' : \prod_{i\in I} \underline{B}_i$ and $\hat{\imath} \in I$ implies $V\hat{\imath}\ \mathcal{R}\ V'\hat{\imath} : \underline{B}_{\hat{\imath}}$
- $M\ \mathcal{R}\ M' : \sum_{i\in I} A_i$ and $M \Downarrow \langle \hat{\imath}, V\rangle$ implies that $M' \Downarrow \langle \hat{\imath}, V'\rangle$ and $V\ \mathcal{R}\ V' : A_{\hat{\imath}}$ for some V'.

[4] In the second line below, we use () for pairing to avoid confusion with the object language.

It is a *lower bisimulation* when both $\mathcal{R}$ and $\mathcal{R}^{\mathsf{op}}$ are lower simulations. The greatest lower simulation is called *lower similarity* ($\lesssim$), and the greatest lower bisimulation is called *lower bisimilarity* ($\simeq$).

Definition 2.4 (i) For a closed term M of type A, we write $b(M)$ for its equivalence class modulo lower bisimilarity.

(ii) We write $[\![A]\!]$ for the class of *semantic nodes* i.e. equivalence classes of closed terms.

(iii) We write $[\![A]\!]_{\leqslant\aleph_0}$ for the set of *countably branching semantic nodes*, i.e. equivalence classes that contain a countably branching term.

Remark 2.5 An equivalence class contains a countably branching term iff it contains a *finitely branching* term (one in which all the nondeterminism cardinals are finite), because $\mathtt{choose}_{n\in\mathbb{N}} M_n$ can be expanded as M_0 `or` $(M_1$ `or` $(\cdots))$ up to lower bisimilarity.

The reason we use the $[\![-]\!]$ notation is that $[\![A]\!]$ is compositional in A. We have isomorphisms

$$\begin{array}{rcl} [\![\textstyle\sum_{i\in I} A_i]\!] & \overset{\beta(\sum_{i\in I} A_i)}{\longrightarrow} & \mathcal{P}\textstyle\sum_{i\in I}[\![A_i]\!] \\ b(M) & \longmapsto & \{(i, b(V)) \mid M \Downarrow \langle i, V\rangle\} \end{array}$$

$$\begin{array}{rcl} [\![\textstyle\prod_{i\in I} \underline{B}_i]\!] & \overset{\beta(\prod_{i\in I} \underline{B}_i)}{\longrightarrow} & \textstyle\prod_{i\in I}[\![\underline{B}_i]\!] \\ b(V) & \longmapsto & i \mapsto b(Vi) \end{array}$$

These restrict to isomorphisms

$$[\![\textstyle\sum_{i\in I} A_i]\!]_{\leqslant\aleph_0} \overset{\beta_{\leqslant\aleph_0}(\sum_{i\in I} A_i)}{\longrightarrow} \mathcal{P}\textstyle\sum_{i\in I}[\![A_i]\!]_{\leqslant\aleph_0}$$

$$[\![\textstyle\prod_{i\in I} \underline{B}_i]\!]_{\leqslant\aleph_0} \overset{\beta_{\leqslant\aleph_0}(\prod_{i\in I} \underline{B}_i)}{\longrightarrow} \textstyle\prod_{i\in I}[\![\underline{B}_i]\!]_{\leqslant\aleph_0}$$

We therefore obtain $[\![A]\!]$ and $[\![A]\!]_{\leqslant\aleph_0}$ compositionally (up to isomorphism) in the case that A is well-founded. This extends to non-well-founded types, as we now explain.

Write types for the set of all types, and define the endofunctor F on $\mathbf{Class}^{\mathsf{types}}$ by

$$(FQ)\textstyle\sum_{i\in I} A_i = \mathcal{P}\sum_{i\in I} Q A_i$$
$$(FQ)\textstyle\prod_{i\in I} \underline{B}_i = \prod_{i\in I} Q\underline{B}_i$$

Then $(\mathsf{Syn}(-), \alpha)$ is a coalgebra for F, and $([\![-]\!], \beta)$ is a final coalgebra. The map $M \mapsto b(M)$ on closed terms is the *anamorphism*, i.e. unique coalgebra morphism to the final coalgebra.

Similarly, we define the endofunctor $F_{\leqslant\aleph_0}$ on $\mathbf{Set}^{\mathsf{types}}$ by

$$(F_{\leqslant\aleph_0}Q)\textstyle\sum_{i\in I} A_i = \mathcal{P}_{\leqslant\aleph_0}\sum_{i\in I} Q A_i$$
$$(F_{\leqslant\aleph_0}Q)\textstyle\prod_{i\in I} \underline{B}_i = \prod_{i\in I} Q\underline{B}_i$$

Then $(\mathsf{Syn}_{\leqslant\aleph_0}(-), \alpha_{\leqslant\aleph_0})$ is a coalgebra for F, and $(\llbracket - \rrbracket_{\leqslant\aleph_0}, \beta_{\leqslant\aleph_0})$ is a final coalgebra. The map $M \mapsto b(M)$ on countably branching closed terms is again the anamorphism.

We use the same notation for semantic nodes as for syntactic nodes.

- For $M \in \llbracket \sum_{i\in I} A_i \rrbracket$ and $\hat{\imath} \in I$ and $V \in \llbracket A_{\hat{\imath}} \rrbracket$, we write $M \Downarrow \langle \hat{\imath}, V \rangle$ when $(\hat{\imath}, V) \in \beta(\sum_{i\in I} A_i)M$.
- For $V \in \llbracket \prod_{i\in I} \underline{B}_i \rrbracket$ and $\hat{\imath} \in I$, we write $V\hat{\imath} \stackrel{\text{def}}{=} (\beta(\prod_{i\in I} \underline{B}_i)V)\hat{\imath}$.

Lemma 2.6 *There exists a function mapping each semantic node $V \in \llbracket A \rrbracket$ to a syntactic node $a(V) \in \mathsf{Syn}(A)$, in such a way that*

- *$V\hat{\imath} \Downarrow \langle \hat{\jmath}, W \rangle$ implies $a(V)\hat{\imath} \Downarrow \langle \hat{\jmath}, a(W) \rangle$*
- *if V is countably branching then $a(V)$ is countably branching.*

Hence, by bisimulation, $b(a(V)) = V$.

Proof. Here is one such function, defined by guarded recursion: if $V \in \llbracket \prod_{i\in I} \sum_{j\in J_i} A_{ij} \rrbracket$ then

$$a(V) = \lambda\{i.\mathtt{choose}^{\mathtt{div}}\{\langle j, a(W)\rangle\}_{Vi\Downarrow\langle j,W\rangle}\}$$

□

We fix such a function for the rest of the paper.

2.3 Open Terms

In order to define the operational meaning of an open term, we must first adapt our existing concepts from nodes to environments.

Definition 2.7 Let Γ be a typing context.

(i) A *syntactic environment* (resp. *semantic environment*) ρ maps each $(\mathtt{x} : A) \in \Gamma$ to a syntactic node (resp. semantic node) of type A.

(ii) We say that a syntactic or semantic environment ρ for Γ is *countably branching* when $\rho(\mathtt{x})$ is countably branching for each $(\mathtt{x} : A) \in \Gamma$.

(iii) We write $\mathsf{Syn}(\Gamma)$ (resp. $\llbracket\Gamma\rrbracket$, $\mathsf{Syn}_{\leqslant\aleph_0}(\Gamma)$, $\llbracket\Gamma\rrbracket_{\leqslant\aleph_0}$) for the set of syntactic (resp. semantic, countably branching syntactic, countably branching semantic) environments for Γ.

(iv) If ρ is a syntactic environment for Γ, we write $b(\rho)$ for the semantic environment $\mathtt{x} \mapsto b(\rho(\mathtt{x}))$.

(v) Let $\Gamma \vdash M : A$ be a term and ρ a syntactic environment for Γ. We write $\vdash M[\rho] : A$ for the closed term obtained by substituting ρ in M.

Proposition 2.8 *For any term $\Gamma \vdash M : A$, the function $\mathsf{Syn}(\Gamma) \longrightarrow \mathsf{Syn}(A)$ mapping $\rho \mapsto M[\rho]$ preserves lower similarity and preserves lower bisimilarity.*

Prop. 2.8 is a consequence of the following.

Lemma 2.9 *For a type-indexed relation $\mathcal{R}$, let $\mathsf{id}[\mathcal{R}]$ be the type-indexed relation given at A by*

$$\{(M[\rho], M[\rho']) \mid \Gamma \vdash M : A, \rho, \rho' \in \mathsf{Syn}(\Gamma) \mid \forall (\mathtt{x} : A) \in \Gamma.\ \rho(\mathtt{x})\ \mathcal{R}\ \rho'(\mathtt{x})\}$$

If $\mathcal{R}$ is a lower simulation, then $\mathsf{id}[\mathcal{R}]$ is a lower simulation.

It is easy to prove Lemma 2.9 by induction on $\Downarrow$, but it is more intuitive to deduce it from Prop. 3.8 below.

We can now speak of the meaning of an open term.

Definition 2.10 Let $\Gamma \vdash M : A$ be a term.

(i) We define the function $[\![\Gamma]\!] \xrightarrow{m(M)} [\![A]\!]$ to map $b(\rho)$ to $b(M[\rho])$. This is well defined by Prop. 2.8.

(ii) We write $[\![\Gamma]\!]_{\leqslant \aleph_0} \xrightarrow{m_{\leqslant \aleph_0}(M)} [\![A]\!]$ for the restriction of $m(M)$ to $[\![\Gamma]\!]_{\leqslant \aleph_0}$. Clearly, if M is countably branching, then we have $[\![\Gamma]\!]_{\leqslant \aleph_0} \xrightarrow{m_{\leqslant \aleph_0}(M)} [\![A]\!]_{\leqslant \aleph_0}$.

Remark 2.11 Two open terms $\Gamma \vdash M, M' : A$ are said to be *lower applicatively bisimilar* [1] when $m(M) = m(M')$. Lower applicative bisimilarity is in fact a congruence, though the proof is nontrivial [2,4]. We do not use that result in this paper, but it does tell us that $m(-)$ is a "reasonable" notion of meaning.

Remark 2.12 We can construct an isomorphism

$$\prod_{i \in I} \sum_{j \in J_i} A_{ij} \cong \prod_{\langle\rangle \in 1} \sum_{\langle i,j \rangle \in \sum_{i \in I} J_i} A_{ij} \tag{1}$$

by means of the following terms

$$\mathtt{x} : \mathrm{LHS} \vdash^{\mathsf{v}} \ \lambda\langle\rangle.\ \mathtt{choose}^{\mathtt{div}} i \in I.\ \mathtt{pm}\ \mathtt{x}\, i\ \mathtt{as}\ \langle j, \mathtt{z}\rangle.\ \langle\langle i, j\rangle, \mathtt{z}\rangle \ : \mathrm{RHS}$$

$$\mathtt{y} : \mathrm{RHS} \vdash^{\mathsf{v}} \quad \lambda i.\ \mathtt{pm}\ \mathtt{y}\langle\rangle\ \mathtt{as} \begin{cases} \langle\langle i, j\rangle, \mathtt{w}\rangle. & \langle j, \mathtt{y}\rangle \\ \langle\langle i' \neq i, j\rangle, \mathtt{w}\rangle.\ \mathtt{diverge} & \end{cases} : \mathrm{LHS}$$

which are inverse up to lower applicative bisimilarity, and hence up to may-testing, cf. [7]. However, up to divergence-sensitive equivalences, such as must testing or convex applicative bisimilarity, (1) is not valid. Moreover, it is invalid in the deterministic setting.

Our goal is to solve the following:

Problem 2.13 *Characterize those functions $[\![\Gamma]\!]_{\leqslant \aleph_0} \xrightarrow{f} [\![A]\!]$ that are* definable *i.e. such that $f = m_{\leqslant \aleph_0}(M)$ for some term $\Gamma \vdash M : A$.*

Note that M may use unrestricted nondeterminism. The following task is also interesting, but remains unsolved:

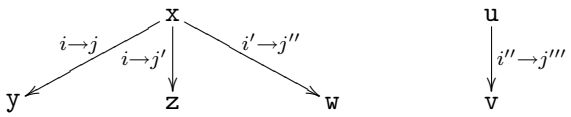
Problem 2.14 *Characterize those functions $[\![\Gamma]\!]_{\leqslant \aleph_0} \xrightarrow{f} [\![A]\!]_{\leqslant \aleph_0}$ that are definable by a countably branching term.*

Although "characterize" is not precisely defined, there is one concrete requirement we impose. If we restrict Problems 2.13–2.14 to the case where A and all the types in Γ are finite, then the two problems coincide (this will be proved later—see end of Sect. 3.4). In this case, we require a decision procedure for the definability of f, and in the case that f is definable, an algorithm that yields a (finitely branching) defining term.

3 Exploration

3.1 Developed Contexts

To solve Problem 2.13, it is helpful to generalize the notion of typing context to include information about how bindings were made. Here is an example of such a "developed" context:



The context contains identifiers $\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{u}, \mathtt{v}$, and any environment ρ for this context is required to satisfy $\rho(\mathtt{x})i \Downarrow \langle j, \rho(\mathtt{y})\rangle$ etc.

Definition 3.1 (i) An *edge* over a typing context Γ_0 is a quadruple $\langle \mathtt{x}, i, j, \mathtt{y}\rangle$ where $(\mathtt{x} : \prod_{i\in I}\sum_{j\in J_i} A_{ij}) \in \Gamma_0$ and $i \in I$ and $j \in J_i$ and $(\mathtt{y} : A_{ij}) \in \Gamma_0$.

(ii) A *developed context* Γ consists of a typing context Γ_0 equipped with a set R of edges over Γ_0 that is forest-structured, i.e.

- for each $\mathtt{y} \in \Gamma_0$ there is at most $\mathtt{x}, i, j$ such that $\langle \mathtt{x}, i, j, \mathtt{y}\rangle \in R$—if there is none, $\mathtt{y}$ is a *root*
- (acyclicity) there is no sequence

$$\mathtt{x}_0, i_0, j_0, \ldots, \mathtt{x}_{n-1}, i_{n-1}, j_{n-1}, \mathtt{x}_n = \mathtt{x}_0$$

such that $n > 0$ and $\langle \mathtt{x}_r, i, j, \mathtt{y}_{r+1}\rangle \in R$ for each $r < n$.

(iii) A *syntactic environment* (resp. semantic environment) ρ for a developed context Γ maps each identifier $(\mathtt{x} : A) \in \Gamma$ to a syntactic node (resp. semantic node) of type A in such a way that $\rho(\mathtt{x})i \Downarrow \langle j, \rho(\mathtt{y})\rangle$ for each edge $\langle \mathtt{x}, i, j, \mathtt{y}\rangle$ of Γ. Def. 2.7 (ii)–(iv) can be applied to developed contexts.

(iv) A *renaming* $\Gamma \xrightarrow{\theta} \Gamma'$ of developed contexts maps each identifier in Γ to an identifier of the same type in Γ' such that roots are mapped to roots and edges to edges. If ρ is a (syntactic or semantic) environment for Γ', we write $\theta^\circ\rho$ for the environment for Γ given by $\mathtt{x} \mapsto \rho(\theta(\mathtt{x}))$.

An ordinary typing context can be seen as a developed context where all the identifiers are roots. Conversely, we can obtain a typing context Γ_0 from a developed context Γ by ignoring the edges, and we write $\Gamma \vdash M : A$ as shorthand for $\Gamma_0 \vdash M : A$.

We note that, if ρ is a semantic environment for a developed context Γ, then $a(\rho)$, defined as $\mathtt{x} \mapsto a(\rho(\mathtt{x}))$, is a syntactic environment for Γ and $b(a(\rho)) = \rho$. Furthermore $a(\rho)$ is countably branching if ρ is.

We can generalize Problem 2.13 to the case that Γ is a developed function. The following terminology is useful.

Definition 3.2 A *developed function* into a type A is a developed context Γ together with a function $[\![\Gamma]\!]_{\leqslant \aleph_0} \xrightarrow{f} [\![A]\!]$.

Thus our task is to identify when a developed function $\langle \Gamma, f \rangle$ into A is definable by some term $\Gamma \vdash M : A$.

3.2 Exploration of a Syntactic Environment

Suppose we have a term $\Gamma \vdash M : A$ and a syntactic environment ρ for Γ. Any evaluation $M[\rho] \Downarrow \langle \hat{\imath}, V \rangle$ must "explore" ρ to a certain finite extent, and our aim is to make this precise (Prop. 3.8 below). The exploration of each value $\rho(\mathtt{x})$ is tracked as follows.

Definition 3.3 Let V be a (syntactic or semantic) node of value type A.

(i) A *node trace* s from V is a finite sequence of tags and nodes

$$i_0, j_0, V_1, \ldots, i_{n-1}, j_{n-1}, V_n \text{ where } V_0 i_0 \Downarrow \langle j_0, V_1 \rangle \cdots V_{n-1} i_{n-1} \Downarrow \langle j_{n-1}, V_n \rangle$$

writing $V_0 \stackrel{\text{def}}{=} V$. The *end-node* of s is V_n and the *end-type* is the type of V_n.

(ii) An *exploration tree* from V is a finite set of node traces that is prefix-closed and contains ε (the empty sequence).

Definition 3.4 Let ρ be a (syntactic or semantic) environment for a developed context Γ.

(i) An *exploration* T of ρ associates to each $(\mathtt{x} : A) \in \Gamma$ an exploration tree $T(\mathtt{x})$ for $\rho(\mathtt{x})$.

(ii) Given an exploration T, we define Γ_T to be the developed context consisting of identifiers $\mathtt{y}_{\mathtt{x},s}$ for each $\mathtt{x} \in \Gamma$ and $s \in T(\mathtt{x})$. The type of $\mathtt{y}_{\mathtt{x},s}$ is the end-type of s. The edges are as follows.
- $\mathtt{y}_{\mathtt{x},\varepsilon}$ is a root of Γ_T, for each root $\mathtt{x}$ of Γ
- $\langle \mathtt{y}_{\mathtt{x},\varepsilon}, i, j, \mathtt{y}_{\mathtt{x}',\varepsilon} \rangle$ is an edge for Γ_T, for each edge $\langle \mathtt{x}, i, j, \mathtt{x}' \rangle$ of Γ
- $\langle \mathtt{y}_{\mathtt{x},s}, i, j, \mathtt{y}_{\mathtt{x},s+(i,j,n)} \rangle$ is an edge for Γ_T, for each $\mathtt{x} \in \Gamma$ and $s + (i, j, n) \in T(\mathtt{x})$

(iii) We define the renaming $\Gamma \xrightarrow{\psi_T} \Gamma_T$ mapping $\mathtt{x}$ to $\mathtt{y}_{\mathtt{x},\varepsilon}$.

(iv) If ζ is a syntactic (resp. semantic) environment for Γ then a *T-descendant* of ζ is a syntactic (resp. semantic) environment ξ of Γ_T such that $\psi_T^\circ \xi = \zeta$.

(v) The *principal* T-descendant η_T of ρ maps $\mathtt{y}_{\mathtt{x},s}$ to the end-node of s.

We see that countability is preserved by descent.

Lemma 3.5 *Let T be an exploration of a (syntactic or semantic) environment ρ for a developed context Γ.*

(i) *If ζ is a countably branching (syntactic or semantic) environment for Γ, then every T-descendant of ζ is countably branching.*

(ii) *If ξ is a countably branching (syntactic or semantic) environment for Γ_T, then ξ is a T-descendant of some countably branching environment ζ for Γ.*

Proof.

(i) For each $\mathtt{y}_{\mathtt{x},s} \in \Gamma_T$, we prove that $\zeta\mathtt{y}_{\mathtt{x},s}$ is countably branching, by induction on s.

(ii) Take $\zeta \stackrel{\text{def}}{=} \psi_T^\circ \xi$.

□

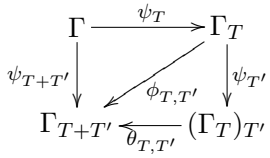
Remark 3.6 Although Lemma 3.5(ii), which is used in the proof of Lemma 3.15, is trivial, we state it explicitly for the sake of Sect. 4, where the analogous result is harder to prove.

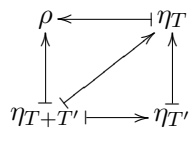
We introduce some notation for explorations, which we shall use in the proof of Prop. 3.8 below.

Definition 3.7 Let Γ be a developed context. Let ρ be a (syntactic or semantic) environment for Γ.

(i) We write ε_ρ for the exploration of ρ that maps $(\mathtt{x} : A) \in \Gamma$ to $\{\varepsilon\}$.

(ii) Let T be an exploration of ρ and T' an exploration of η_T (as an environment for Γ_T). We define $T + T'$ to be the exploration of ρ mapping $(\mathtt{x} : A) \in \Gamma$ to $\{s + s' \mid s \in T(\mathtt{x}), s' \in T'(\mathtt{y}_{\mathtt{x},s})\}$. Then we write $\Gamma_T \xrightarrow{\phi_{T,T'}} \Gamma_{T+T'}$ for the inclusion renaming, and $(\Gamma_T)_{T'} \xrightarrow{\theta_{T,T'}} \Gamma_{T+T'}$ for the renaming $\mathtt{y}_{\mathtt{y}_{\mathtt{x},s},s'} \mapsto \mathtt{y}_{\mathtt{x},s+s'}$. So we have a commutative diagram of renamings





These act on the special environments as shown on the right.

(iii) If T is an exploration of ρ, and $(\mathtt{x} : A) \in \Gamma$, and s is a node-trace of $T(\mathtt{x})$ with end-node V and $Vi \Downarrow \langle j, W\rangle$, then we define $T(\mathtt{x}, s + (i, j, W))$ to be the exploration on ρ given by

$$\begin{cases} \mathtt{x} \mapsto T(\mathtt{x}) \cup \{s + (i, j, W)\} & \\ \mathtt{y} \mapsto T(\mathtt{y}) & \text{if } \mathtt{y} \neq \mathtt{x} \end{cases}$$

We now give the key operational result.

$$\frac{}{\Gamma; \langle \hat{\imath}, V\rangle; \rho \downarrow \hat{\imath}; \epsilon_\rho; V[\psi_{\epsilon_\rho}]} \qquad \frac{\Gamma; M; \rho \downarrow \hat{\imath}; T_0; W \quad \Gamma_T; N_{\hat{\imath}}[\psi_T, W/\mathtt{x}]; \eta_T \downarrow \hat{\jmath}; T_1'; W'}{\Gamma; \mathtt{pm}\ M\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\ N_i\}_{i\in I}; \rho \downarrow \hat{\jmath}; T_0 + T_1'; W'[\theta_{T,T'}]}$$

$$\frac{\Gamma; M_{\hat{\imath}}; \rho \downarrow \hat{\jmath}; T; W}{\Gamma; \lambda\{i.\ M_i\}_{i\in I}\, \hat{\imath}; \rho \downarrow \hat{\jmath}; T; W} \qquad \frac{}{\Gamma; \mathtt{x}\,\hat{\imath}; \rho \downarrow \hat{\jmath}; \varepsilon_\rho(\mathtt{x}, (\hat{\imath}, \hat{\jmath}, V)); \mathtt{y}_{\mathtt{x},(\hat{\jmath},\hat{\imath},V)}}\ \rho(\mathtt{x})\hat{\imath} \Downarrow \langle \hat{\jmath}, V\rangle$$

$$\frac{\Gamma; M_{\hat{\jmath}}; \rho \downarrow \hat{\jmath}; T; W}{\Gamma; \mathtt{choose}_{\ j<\alpha}\ M_j; \rho \downarrow \hat{\jmath}; T; W}\ \hat{\jmath} < \alpha$$

Fig. 3. Inductive definition of "causal convergence" relation $\downarrow$, used in proof of Prop. 3.8

Proposition 3.8 *(Syntactic exploration) Let Γ be a developed*[5] *context, let $\Gamma \vdash^{\mathsf{c}} M : \sum_{i\in I} A_i$ be a term and let ρ be a syntactic environment for Γ. If $M[\rho] \Downarrow \langle \hat{\imath}, V\rangle$ then there exists*

- *an exploration T of ρ*
- *a value $\Gamma_T \vdash^{\mathsf{v}} W : A_{\hat{\imath}}$*

such that

- $V = W[\eta_T]$
- *for each syntactic environment ζ of Γ and each T-descendant ξ of ζ, we have $M[\zeta] \Downarrow \langle \hat{\imath}, W[\xi]\rangle$.*

If M and ρ are countably branching, then so is W.

Proof. We define a predicate $\Gamma; M; \rho \downarrow \hat{\imath}; T; W$ where

- $\Gamma \vdash^{\mathsf{c}} M : \sum_{i\in I} A_i$ and ρ is a syntactic environment for Γ
- $\hat{\imath} \in I$ and T is an exploration of ρ and $\Gamma_T \vdash^{\mathsf{v}} W : A_{\hat{\imath}}$

This is called "causal convergence" because it indicates the exploration that caused a convergence to happen. It is defined inductively in Fig. 3. Note that if $\Gamma; M; \rho \downarrow \hat{\imath}; T; W$, and M and ρ are both countably branching, then so is W.

We prove by induction that if $\Gamma \vdash^{\mathsf{c}} M : \sum_{i\in I} A_i$ and ρ is a syntactic environment for Γ and $M[\rho] \Downarrow \langle \hat{\imath}, V\rangle$ then there exists an exploration T of ρ and a value $\Gamma_T \vdash^{\mathsf{v}} W : A_{\hat{\imath}}$ such that $\Gamma; M; \rho \downarrow \hat{\imath}; T; W$ and $W[\eta_T] = V$. The inductive step depends on the form of M—we omit details.

Next, we prove by induction that if $\Gamma; M; \rho \downarrow \hat{\imath}; T; W$, then, for any syntactic environment ζ of Γ and any T-descendant ξ of ζ, we have $M[\zeta] \Downarrow \langle \hat{\imath}, W[\xi]\rangle$—this will complete our proof. The inductive step depends on the form of M—we omit details. □

[5] The result is no weaker if we say that Γ is a typing context, but the "developed" formulation is more convenient for the sequel.

3.3 Exploration of a Semantic Environment

In Prop. 3.10 below, we present properties of developed functions that are definable, both into a value type and into a computation type. Each of these requires a definition, formulated as follows.

Definition 3.9 (i) Let $\langle \Gamma, f \rangle$ be a developed function into $\prod_{i \in I} \underline{B}_i$. For $\hat{\imath} \in I$, we write $[\![\Gamma]\!]_{\leqslant \aleph_0} \xrightarrow{f_{\hat{\imath}}} \underline{B}_{\hat{\imath}}$ to be $\rho \mapsto f(\rho)\hat{\imath}$.

(ii) Let $p = \langle \Gamma, f \rangle$ be a developed function into $\sum_{i \in I} A_i$.

(a) A *convergence datum* $q = \langle \rho, \hat{\imath}, V \rangle$ for p consists of a semantic environment $\rho \in [\![\Gamma]\!]_{\leqslant \aleph_0}$, a tag $\hat{\imath} \in I$ and an element $V \in [\![A_{\hat{\imath}}]\!]$ such that $f(\rho) \Downarrow \langle \hat{\imath}, V \rangle$.

(b) A *cause* of a convergence datum $q = \langle \rho, \hat{\imath}, V \rangle$ for p consists of an exploration T of ρ and a function $[\![\Gamma_T]\!]_{\leqslant \aleph_0} \xrightarrow{g} [\![A_{\hat{\imath}}]\!]$ such that

- $g(\eta_T) = b$
- for any $\zeta \in [\![\Gamma]\!]_{\leqslant \aleph_0}$ and any T-descendant ξ of ζ, we have $f(\zeta) \Downarrow \langle \hat{\imath}, g(\xi) \rangle$.

Proposition 3.10 (i) *Let $\langle \Gamma, f \rangle$ be a developed function into $\prod_{i \in I} \underline{B}_i$ that is definable. Then for every $\hat{\imath} \in I$ the developed function $\langle \Gamma, f_{\hat{\imath}} \rangle$ into $\underline{B}_{\hat{\imath}}$ is definable.*

(ii) *Let $p = \langle \Gamma, f \rangle$ be a developed function into $\sum_{i \in I} A_i$ that is definable. Then every convergence datum $q = \langle \rho, \hat{\imath}, V \rangle$ of p has a cause $\langle T, g \rangle$ such that the developed function $\langle \Gamma_T, g \rangle$ into $A_{\hat{\imath}}$ is definable.*

Proof.

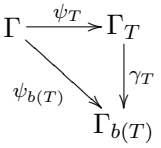
(i) If $\langle \Gamma, f \rangle$ is defined by M, then $\langle \Gamma, f_{\hat{\imath}} \rangle$ is defined by $M\hat{\imath}$.

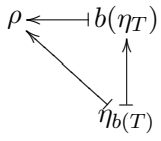
(ii) Let p be defined by M, and let $q = \langle \rho, \hat{\imath}, V \rangle$ be a convergence datum of p. Then

$$b(M[a(\rho)]) = m(M)b(a(\rho)) = m(M)\rho \Downarrow \langle \hat{\imath}, V \rangle$$

so there is a syntactic node $U \in \mathsf{Syn}(A_{\hat{\imath}})$ such that $b(U) = V$ and $M[a(\rho)] \Downarrow \langle \hat{\imath}, U \rangle$.

We then obtain T and W following Prop. 3.8. We obtain an exploration $b(T)$ of ρ mapping $\mathtt{x} \mapsto \{b(s) \mid s \in T\}$. Here $b(s)$ is defined by replacing each syntactic node V in s with $b(V)$. We define a renaming $\Gamma_T \xrightarrow{\gamma_T} \Gamma_{b(T)}$ mapping $\mathtt{y}_{\mathtt{x},s} \mapsto \mathtt{y}_{\mathtt{x},b(s)}$, giving a commutative diagram of renamings





These act on the special environments as shown on the right. Let g be the composite

$$[\![\Gamma_{b(T)}]\!]_{\leqslant \aleph_0} \xrightarrow{\gamma_T^\circ} [\![\Gamma_T]\!]_{\leqslant \aleph_0} \xrightarrow{m(W)} [\![A_{\hat{\imath}}]\!]_{\leqslant \aleph_0}$$

It is then easily checked that $\langle b(T), g \rangle$ is a cause of q.

□

Prop. 3.10 indicate the following coinductive concept.

Definition 3.11 Let E be a predicate on developed functions. We define another predicate ΦE on developed functions as follows.

- $p = \langle \Gamma, f \rangle$ into $\prod_{i \in I} \underline{B}_i$ satisfies ΦE when $\langle \Gamma, f_{\hat{\imath}} \rangle$ into $\underline{B}_{\hat{\imath}}$ satisfies E for each $\hat{\imath} \in I$.
- $p = \langle \Gamma, f \rangle$ into $\sum_{i \in I} A_i$ satisfies ΦE when each convergence datum $q = \langle \rho, \hat{\imath}, V \rangle$ of p has a cause $\langle T, g \rangle$ such that $\langle \Gamma_T, g \rangle$ into $A_{\hat{\imath}}$ satisfies E.

A postfixed point of Φ is called an *exploration predicate*, and the largest one is called *exploratoriness*.

Proposition 3.12 *Let $p = \langle \Gamma, f \rangle$ be a developed function into A. If p is definable, then it is exploratory.*

Proof. By Prop. 3.10, definability is an exploration predicate. □

As usual, the coinductive definition can be formulated in terms of a two-player (Proponent/Opponent) game, with Opponent moving first. Whenever it is Opponent's turn to play, there is a developed function into some type on the table.

- If $p = \langle \Gamma, f \rangle$ into $\prod_{i \in I} \underline{B}_i$ is on the table, then Opponent chooses some $\hat{\imath} \in I$. Proponent replies "continue", and the game continues with $\langle \Gamma, f_{\hat{\imath}} \rangle$ into $\underline{B}_{\hat{\imath}}$ on the table.
- If $p = \langle \Gamma, f \rangle$ into $\sum_{i \in I} A_i$ is on the table, then Opponent chooses some convergence datum $q = \langle \rho, \hat{\imath}, V \rangle$ for p. Proponent replies by choosing a cause $\langle T, g \rangle$ of q, and the game continues with $\langle \Gamma_T, g \rangle$ into $A_{\hat{\imath}}$ on the table.

We call this the "exploration game". A developed function p into A is exploratory iff, beginning the game with p on the table, there exists a strategy for Proponent. (That is, a strategy enabling Proponent to keep playing forever, no matter how Opponent plays.)

The fact that exploratoriness is a fixed point of Φ means that, for well-founded types A, exploratoriness can be given by induction on A. In particular, for developed functions into finite type, we immediately obtain a decision procedure for exploratoriness.

3.4 Exploratoriness Implies Definability

We next see that the converse to Prop. 3.12 is true, by exploiting the unrestricted nondeterminism in the language.

Proposition 3.13 *Let $p = \langle \Gamma, f \rangle$ be a developed function into A. If p is exploratory, then it is definable.*

Proof. Firstly, given an exploration T of $\rho \in [\![\Gamma]\!]$ and a computation $\Gamma_T \vdash^{\mathsf{c}} M : \underline{B}$,

we define a computation $\Gamma \vdash^{\mathsf{c}} T^*M : \underline{B}$ such that

$$m(T^*M) : \zeta \mapsto \bigcup_{\xi \in \mathsf{Trav}(T,\zeta)} [M]\xi \qquad \text{for any } \zeta \in [\![\Gamma]\!] \tag{2}$$

where $\mathsf{Trav}(T, \zeta)$ is the set of T-descendants of ζ. We define T^*M by induction on T (using some choice).

- If $T = \varepsilon_\rho$, then ψ_T is a bijection and we define $T^*M \stackrel{\text{def}}{=} M[\psi_T^{-1}]$.
- Otherwise $T = T'(\mathtt{x}, s(i, j, V))$ where T' is smaller than T, and we define T^*M to be

$$T'^* \left(\mathtt{pm}\ \mathtt{y}_{\mathtt{x},s'}\hat{\imath}\ \mathtt{as} \begin{cases} \langle \hat{\jmath}, \mathtt{y}_{\mathtt{x},s} \rangle . \quad M \\ \langle j \neq \hat{\jmath}, \mathtt{y} \rangle .\ \mathtt{diverge} \end{cases} \right)$$

  which satisfies (2) by

$$m(T^*M)\zeta = \bigcup_{\xi \in \mathsf{Trav}(T',\zeta)} \bigcup_{\langle \hat{\jmath}, c \rangle \in (\zeta(\mathtt{y}_{\mathtt{x},s'}))\hat{\imath}} m(M)(\xi, \mathtt{y}_{\mathtt{x},s} \mapsto c) = \bigcup_{\xi' \in \mathsf{Trav}(T,\zeta)} m(M)\xi'$$

Secondly, by the axiom of choice, there is a function mapping each convergence datum $q = \langle \rho, \hat{\imath}, a \rangle$ of each exploratory developed function $p = \langle \Gamma, f \rangle$ into $\sum_{i \in I} A_i$ to a cause $\langle T_q, g_q \rangle$ of q such that $\langle \Gamma_{T_q}, g_q \rangle$ into $A_{\hat{\imath}}$ is exploratory. We then have, for each exploratory developed function $p = \langle \Gamma, f \rangle$ into $\sum_{i \in I} A_i$ and each $\zeta \in [\![\Gamma]\!]_{\leqslant \aleph_0}$,

$$f(\zeta) = \{ \langle i, g_q(\xi) \rangle \mid q = \langle \rho, i, a \rangle \in c(p), \xi \in \mathsf{Trav}(T_q, \zeta) \} \tag{3}$$

To prove (3), we see that $\supseteq$ is immediate from the definition of "cause". For $\subseteq$, if $\langle \hat{\imath}, b \rangle \in$ LHS, then $\langle \hat{\imath}, b \rangle \in$ RHS putting $\rho \stackrel{\text{def}}{=} \zeta$, $i \stackrel{\text{def}}{=} \hat{\imath}$, $a \stackrel{\text{def}}{=} b$ and $\xi \stackrel{\text{def}}{=} \eta_{T_q}$.

For each exploratory developed function $p = \langle \Gamma, f \rangle$ into type A, we define $\Gamma \vdash t_A p : A$ by guarded recursion as follows:

$$t_{\prod_{i \in I} \underline{B}_i} \langle \Gamma, f \rangle \stackrel{\text{def}}{=} \lambda i \in I . t_{\underline{B}_i} \langle \Gamma, f_i \rangle$$

$$t_{\sum_{i \in I} A_i} \langle \Gamma, f \rangle \stackrel{\text{def}}{=} \mathtt{choose}^{\mathtt{div}} \{ T_q^* \langle i, t_{A_i} \langle \Gamma_{T_q}, g_q \rangle \rangle \}_{q = \langle \rho, i, a \rangle \in c(\langle \Gamma, f \rangle)}$$

where $c(p)$ is the set of convergence data of p.

We have (omitting the β isomorphisms)

$$m(t_{\prod_{i \in I} \underline{B}_i} \langle \Gamma, f \rangle)\rho = \lambda i \in I . m(t_{\underline{B}_i} \langle \Gamma, f_i \rangle)\rho \tag{4}$$

$$\begin{aligned} m(t_{\sum_{i \in I} A_i} \langle \Gamma, f \rangle)\zeta &= \bigcup_{q = \langle \rho, i, a \rangle \in c(\langle \Gamma, f \rangle)} \bigcup_{\xi \in \mathsf{Trav}(T_q, \zeta)} \{ \langle i, m(t_{A_i} \langle \Gamma_{T_q}, g_q \rangle)\xi \rangle \} \\ &= \{ \langle i, m(t_{A_i} \langle \Gamma_{T_q}, g_q \rangle)\xi \rangle \mid q = \langle \rho, i, a \rangle \in c(\langle \Gamma, f \rangle), \xi \in \mathsf{Trav}(T_q, \zeta) \} \end{aligned}$$

Let $\mathcal{R}$ be the type-indexed relation that relates $m(t_A \langle \Gamma, f \rangle)\rho$ to $f(\rho)$ at type A, for each exploratory developed function $p = \langle \Gamma, f \rangle$ into A and each $\rho \in [\![\Gamma]\!]_{\leqslant \aleph_0}$. We show that $\mathcal{R}$ is a bisimulation, so that it is included in the identity i.e. $m(t_A p)\rho = f(\rho)$.

- Suppose $x \mathcal{R} y : \prod_{i\in I}\underline{B}_i$. Then $x = m(t_{\prod_{i\in I}\underline{B}_i}p)\rho$ and $y = f(\rho)$ for some exploratory developed function $p = \langle\Gamma, f\rangle$ into $\prod_{i\in I}\underline{B}_i$, and some $\rho \in [\![\Gamma]\!]_{\leqslant\aleph_0}$. Then for each $\hat{\imath} \in I$ equation (4) gives us

$$x\,\hat{\imath} = m(t_{\underline{B}_{\hat{\imath}}}\langle\Gamma, f_{\hat{\imath}}\rangle)\rho \mathcal{R} f_{\hat{\imath}}(\rho) = y\,\hat{\imath}$$

- Suppose $x \mathcal{R} y : \sum_{i\in I}A_i$. Then $x = m(t_{\sum_{i\in I}A_i}p)\rho$ and $y = f(\rho)$ for some exploratory developed function $p = \langle\Gamma, f\rangle$ into $\sum_{i\in I}A_i$, and some $\rho \in [\![\Gamma]\!]_{\leqslant\aleph_0}$.
  - If $x \Downarrow \langle\hat{\imath}, x'\rangle$, then by (5) $x' = m(t_{A_i}\langle\Gamma_{T_q}, g_q\rangle)\xi$ for some $q = \langle\rho, i, a\rangle \in c(\langle\Gamma, f\rangle)$ and $\xi \in \mathsf{Trav}(T_q, \zeta)$. Put $y' \stackrel{\text{def}}{=} g_q(\xi)$ giving $x' \mathcal{R} y'$ and by (3) $y \Downarrow \langle\hat{\imath}, y'\rangle$.
  - If $y \Downarrow \langle\hat{\imath}, y'\rangle$ then by (3) $y' = g_q(\xi)$ for some $q = \langle\rho, i, a\rangle \in c(\langle\Gamma, f\rangle)$ and $\xi \in \mathsf{Trav}(T_q, \zeta)$. Put $x' = m(t_{A_i}\langle\Gamma_{T_q}, g_q\rangle)\xi$ giving $x' \mathcal{R} y'$ and by (5) $x \Downarrow \langle\hat{\imath}, x'\rangle$.

□

We are particularly interested in the following functions.

Definition 3.14 A developed function $\langle\Gamma, f\rangle$ into A is *countably branching* when the range of f is contained in $[\![A]\!]_{\leqslant\aleph_0}$. We thus have $[\![\Gamma]\!]_{\leqslant\aleph_0} \xrightarrow{f} [\![A]\!]_{\leqslant\aleph_0}$.

The definition of exploratoriness is not changed if we restrict to countably branching functions, because the countable branching property is an *invariant* of the exploration game, in the following sense.

Lemma 3.15 (i) *If $p = \langle\Gamma, f\rangle$ into $\prod_{i\in I}\underline{B}_i$ is countably branching, then so is $\langle\Gamma, f_{\hat{\imath}}\rangle$ into $\underline{B}_{\hat{\imath}}$ for each $\hat{\imath} \in I$.*

(ii) *If $p = \langle\Gamma, f\rangle$ into $\sum_{i\in I}A_i$ is countably branching, then so is $\langle\Gamma_T, g\rangle$ into $A_{\hat{\imath}}$, for each cause $\langle T, g\rangle$ of a convergence datum $q = \langle\rho, \hat{\imath}, V\rangle$ of p.*

Proof. (i) is trivial. For (ii), for $\xi \in [\![\Gamma]\!]_{\leqslant\aleph_0}$, Lemma 3.5(ii) gives us $\zeta \in [\![\Gamma]\!]_{\leqslant\aleph_0}$ such that ξ is a T-descendant of some ζ. So $f(\zeta) \Downarrow \langle\hat{\imath}, g(\xi)\rangle$, and because $f(\zeta)$ is countably branching, $g(\xi)$ is too. □

Let $p = \langle\Gamma, f\rangle$ be a developed function into A. We know that if p definable by a countably branching term, then p is countably branching and exploratory. One might conjecture that if p is countably branching and exploratory, then it is definable by a countably branching term. But that is not so.

Proposition 3.16 *Let Γ be the context $\mathtt{x} : \prod\{*.\sum_{i\in\mathbb{N}}\prod\{\}\}$, and let $\underline{B}$ be the type $\sum\{*.\prod\{*.\sum\{*.\prod\{\}\}\}\}$.*

(i) *There are only $2^{\aleph_0}$ countably branching terms $\Gamma \vdash M : \underline{B}$.*

(ii) *There are $2^{2^{\aleph_0}}$ countably branching developed functions $[\![\Gamma]\!]_{\leqslant\aleph_0} \xrightarrow{f} [\![\underline{B}]\!]_{\leqslant\aleph_0}$ that are exploratory.*

Proof. (i) is obvious.

For (ii), we first show that the set Q of upper subsets of $\mathcal{P}\mathbb{N}$ has size $2^{2^{\aleph_0}}$. Let Q' be the set of $P \subseteq \mathcal{P}\mathbb{N}$ that are *antichains* i.e. such that $A, B \in P$ and

$A \subseteq B$ implies $A = B$. The upper-closure function $Q' \xrightarrow{f} Q$ mapping P to $\{B \mid \exists A \in P. A \subseteq B\}$ is a bijection; its inverse maps $P \in Q$ to the set of minimal elements of $f(P)$. Finally, we define an injection $\mathcal{P}\mathcal{P}\mathbb{N} \xrightarrow{g} Q'$ mapping P to the set $\{\{2n \mid n \in A\} \cup \{2n+1 \mid n \notin A\} \mid A \in P\}$. This gives $2^{2^{\aleph_0}} \leqslant |Q'| \leqslant |Q|$.

For each upper set $P \subseteq \mathcal{P}\mathbb{N}$, we define a term

$$\Gamma \vdash^{\mathsf{c}} M_P \stackrel{\text{def}}{=} \mathtt{choose}\ A \in P.\ \langle *, \lambda\{*.\mathtt{choose}\ n \in A.\ \mathtt{pm}\ (\mathtt{x}*)\ \mathtt{as}\ \langle n, \mathtt{y}\rangle.\ \langle *, \lambda\{\}\rangle\}\rangle : \underline{B}$$

We note that $\{\langle *, \{\}\rangle\} \in m(M_P)(\mathtt{x} \mapsto B)$ iff there exists $A \in P$ that is disjoint from B, i.e. (since P is upper) iff $\mathbb{N} \setminus B \in P$. So we can recover P from $m(M_P)$ via

$$P = \{C \subseteq \mathbb{N} \mid \{\langle *, \{\}\rangle\} \in m(M_P)(\mathtt{x} \mapsto \mathbb{N} \setminus C)\}$$

So $P \neq P'$ implies $m(M_P) \neq m(M_{P'})$, and the set $\{m(M_P) \mid P \in Q\}$ of exploratory functions has cardinality $2^{2^{\aleph_0}}$. □

So if $p = \langle \Gamma, f\rangle$ is a developed function into A that is countably branching and exploratory, how much nondeterminism do we require in the language to ensure that p is definable? It turns out that $2^{\aleph_0}$ is a sufficient cardinal.

Definition 3.17 A term M is *continuum branching* when every nondeterminism cardinal in M is $\leqslant 2^{\aleph_0}$.

Proposition 3.18 *Let $p = \langle \Gamma, f\rangle$ be a countably branching developed function into A. If p is exploratory, then it is definable by a continuum branching term.*

Proof. We note firstly that for any type A, the set $[\![A]\!]_{\leqslant \aleph_0}$ has cardinality $\leqslant 2^{\aleph_0}$, because there are $2^{\aleph_0}$ countably branching terms. Hence $[\![\Gamma]\!]_{\leqslant \aleph_0}$ has cardinality $\leqslant 2^{\aleph_0}$, for any developed context Γ.

Using this fact, and Lemma 3.15, we associate to each exploratory, countably branching developed function $p = \langle \Gamma, f\rangle$ into type A a continuum branching term $\Gamma \vdash t_A p : A$. This is done by guarded recursion as in the proof of Prop. 3.13, and the rest of the proof is the same. □

It is easy to see that, if Γ and A are finite types, then the defining terms exhibited in the proof of Prop. 3.13 are well-founded and finitely branching; and the definition tells us how to obtain the term effectively from an exploratory function.

4 Affine Strategy Calculus

The affine strategy calculus is the same as the intuitionistic version, except that the pm rule in Fig. 1 is replaced by the following.

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \sum_{i \in I} A_i \quad \Gamma', \mathtt{x} : A_i \vdash^{\mathsf{c}} N_i : \underline{B}\ \ (\forall i \in I)}{\Gamma, \Gamma' \vdash^{\mathsf{c}} \mathtt{pm}\ M\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.M_i\}_{i \in I} : \underline{B}}$$

In the affine calculus, if we pattern-match $\mathtt{x}\,i$, obtaining $\langle j, \mathtt{y}\rangle$, then we can pattern-match $\mathtt{y}\,k$ but not $\mathtt{x}\,i'$. Conceptually, the argument is a black box which we access

by playing against, and $\mathtt{y}$ is its current state. We cannot rewind to the earlier state $\mathtt{x}$. This understanding of affineness, closely related to process calculus, follows [7].

We now consider Problem 2.13 in the affine setting. We do not use developed contexts, as new identifiers simply replace old ones. A *function into* a type A is a pair $\langle \Gamma, f \rangle$, where Γ is a typing context and $[\![\Gamma]\!]_{\leqslant \aleph_0} \xrightarrow{f} A$.

To describe exploration of an environment, we do not need an exploration tree for each identifier, just a single node-trace. So we replace Def. 3.4 with the following.

Definition 4.1 (cf. Def. 3.4) Let ρ be a (syntactic or semantic) environment for a typing context Γ.

(i) An *affine exploration* T of ρ associates to each $(\mathtt{x} : A) \in \Gamma$ a node-trace from $\rho(\mathtt{x})$.

(ii) Given an affine exploration T, we define the typing context Γ_T to have the same identifiers as Γ. The type of $\mathtt{x}$ is the end-type of $T(\mathtt{x})$.

(iii) If ζ is a syntactic (resp. semantic) environment for Γ then a *T-descendant* of ζ is a syntactic (resp. semantic) environment ξ of Γ_T such that, for each $(\mathtt{x} : A) \in \Gamma$, there is some node-trace from $\zeta(\mathtt{x})$ to $\xi(\mathtt{x})$ with the same node-erasure as $T(\mathtt{x})$.

(iv) The *principal* T-descendant η_T of ρ maps $\mathtt{x}$ to the end-node of $T(\mathtt{x})$.

We replace Def. 3.7 with the following.

Definition 4.2 Let Γ be a typing context. Let ρ be a (syntactic or semantic) environment for Γ.

(i) We write ε_ρ for the affine exploration of ρ that maps $(\mathtt{x} : A) \in \Gamma$ to $\{\varepsilon\}$.

(ii) Let T be an affine exploration of ρ and T' an affine exploration of η_T (as an environment for Γ_T). We define $T + T'$ to be the affine exploration of ρ mapping $(\mathtt{x} : A) \in \Gamma$ to the node-trace $T(\mathtt{x}) + T'(\mathtt{x})$. So we have a commutative diagram of renamings

$$\begin{array}{ccc} \Gamma & \xrightarrow{\psi_T} & \Gamma_T \\ {\scriptstyle \psi_{T+T'}}\downarrow & & \downarrow{\scriptstyle \psi_{T'}} \\ \Gamma_{T+T'} & = & (\Gamma_T)_{T'} \end{array} \qquad \begin{array}{ccc} \rho & \longleftarrow & \eta_T \\ \uparrow & & \uparrow \\ \eta_{T+T'} & = & \eta_{T'} \end{array}$$

These act on the special environments as shown on the right.

We define $f_{\hat{\imath}}$ and *convergence datum* and *affine cause* as in Def. 3.9, replacing "developed function" with "function" and "exploration" with "affine exploration". A predicate E on functions into types is an *affine exploration predicate* when it satisfies the conditions of Def. 3.11, replacing "cause" with "affine cause". *Affine exploratoriness* is the greatest affine exploration predicate.

All the results of Sect. 2–3 go through with these changes. The proofs are similar, with the exception of Lemma 3.5(ii), which becomes more difficult. In the syntactic case, it suffices to prove that, for any node-trace s from $V \in \mathsf{Syn}(A)$ with end-type

B, and any $W \in \mathsf{Syn}_{\leqslant \aleph_0}(B)$, there is a $U \in \mathsf{Syn}_{\leqslant \aleph_0}(A)$ and a node-trace t from U to W with the same node-erasure as s. This is by induction on s. The semantic case is similar.

5 Further Work

There are several directions in which this work can be developed.

- We can consider *convex bisimilarity*, where divergence is taken into account. In this setting, the intuitionistic calculus takes several forms, depending on whether parallel operators or McCarthy's amb are to be included.
- We can incorporate function types. Is lower applicative bisimilarity then decidable at finite type? In the deterministic setting, the answer is yes if the language is parallel, since the domain model is fully abstract [8,9], and no if the language is sequential [6]. In the nondeterministic setting, the question is open.
- We can try to formulate a denotational model that equates two terms precisely when they are lower applicatively bisimilar. This requires a fixpoint theory for recursion. Our result provides a useful first step, by identifying which functions need to be included in the model.

References

[1] Abramsky, S., *The lazy λ-calculus*, in: *Research topics in Functional Programming*, Addison Wesley, 1990 pp. 65–117.

[2] Howe, D. J., *Proving congruence of bisimulation in functional programming languages*, Inf. and Comp. **124** (1996).

[3] Kahn, G. and G. D. Plotkin, *Concrete domains*, Theoretical Comp. Sci. **121** (1993).

[4] Levy, P. B., *Infinitary Howe's method*, in: *Proc., 8th Intl. Workshop on Coalgebraic Methods in Comp. Sci., Vienna*, ENTCS **164(1)**, 2006.

[5] Levy, P. B., *Infinite trace equivalence*, Annals of Pure & Applied Logic **151** (2008).

[6] Loader, R., *Finitary PCF is not decidable*, TCS **266** (2001), pp. 341–364.

[7] Nygaard, M. and G. Winskel, *Domain theory for concurrency*, TCS **316** (2004).

[8] Plotkin, G. D., *LCF considered as a programming language*, Theoretical Computer Science **5** (1977), pp. 223–255.

[9] Streicher, T., *A universality theorem for PCF with recursive types, parallel-or and exists*, Mathematical Structures in Computer Science **4** (1994), pp. 111–115.