Computational Ludics

Kazushige Terui

Research Institute for Mathematical Sciences, Kyoto University.
Kitashirakawa Oiwakecho, Sakyo-ku, Kyoto 606-8502, JAPAN.

Abstract

We reformulate the theory of ludics introduced by J.-Y. Girard from a computational point of view. We introduce a handy term syntax for designs, the main objects of ludics. Our syntax also incorporates explicit cuts for attaining computational expressivity. We also consider design generators that allow for finite representation of some infinite designs. A normalization procedure in the style of Krivine's abstract machine directly works on generators, giving rise to an effective means of computation over infinite designs.

The acceptance relation between machines and words, a basic concept in computability theory, is well expressed in ludics by the orthogonality relation between designs. Fundamental properties of ludics are then discussed in this concrete context. We prove three characterization results that clarify the computational powers of three classes of designs. (i) Arbitrary designs may capture arbitrary sets of finite data. (ii) When restricted to finitely generated ones, designs exactly capture the recursively enumerable languages. (iii) When further restricted to cut-free ones as in Girard's original definition, designs exactly capture the regular languages.

We finally describe a way of defining data sets by means of logical connectives, where the internal completeness theorem plays an essential role.

Key words: Ludics, automata theory

1 Introduction

Ludics has been introduced by J.-Y. Girard [9] as a foundational, pre-logical framework upon which ordinary logics and type systems are to be built, and in which various semantic and computational phenomena are uniformly analyzed (see [4,2] for good expositions). The basic entities of ludics are called *designs*,

Email address: `terui@kurims.kyoto-u.ac.jp` (Kazushige Terui).

which may be understood in various ways: as abstract sequent proofs, abstract Böhm trees [1], innocent strategies [5] and processes [7]. Ludics then provides a 'forum,' in which various participants (designs) interact together via normalization/composition, and sometimes form a 'community' that shares a common interactive behaviour. Such a 'community' is in fact called a *behaviour* (or an *interactive type*), and corresponds to semantic types (see, e.g., [17]) or truth values in realizability [14]. Ludics sheds a new light onto some known properties, such as confluence/associativity, stability and syntax-semantics correspondence. It also discovers a number of new phenomena, such as incarnation and internal completeness.

Some of the new ideas from ludics are also relevant for the traditional theory of computability and complexity:

Monism. There is no ontological distinction between syntax and semantics. Such a monistic framework would be appealing in the computability theory too, where people usually go back and forth between two ontological entities: *machines* (algorithms) and *languages* (sets), that can be cumbersome. Ludics could provide a forum in which languages and machines are homogeneous entities, only distinguished by their inherent properties. Typically, acceptance relation between machines and words is replaced by the orthogonality relation between designs, which is homogeneous:

$$\text{Machine } \mathsf{M} \text{ accepts a word } w \iff \mathsf{M}^\star \perp w^\star.$$

Focalization. Logical connectives of the same polarity combine together, yielding *synthetic connectives.* After maximal focalization, every logical formula becomes a pure alternation of positive and negative layers. This alternation would give a logical account to the unit of computation time/space (suggested by [9]).

Interaction. As with linear logic and game semantics, ludics favours an interactive view of computation (agent $\leftrightarrow$ agent) rather than the functional one (input $\to$ output). Interactive computation also lies in the core of the basic complexity theory (think of composition of two logspace Turing machines, that has to be done interactively, not functionally), and is also a key notion in the progress in the last two decades (typically in *interactive proof systems*; see e.g. [3]).

Our ultimate goal is to develop a monistic, logical, interactive theory for computability and complexity based on ludics. The current article is a first step towards this goal. We propose a slightly modified and extended formalism for ludics that is well suited for dealing with computational objects. The major modifications are as follows:

(1) Designs in [9] are built with absolute addresses (sequences of natural numbers), called *loci*. While this locative approach is illuminating in theory, it

is too heavy for practical use; working with absolute addresses is like programming with machine codes. We therefore adopt a more conservative approach using a term calculus, where absolute addresses are replaced by variable bindings, as initiated by Curien [2]. Coinductive techniques turn out useful for manipulating our syntactic designs (cf. [13]; see [12] for an introduction to coinductive techniques).

(2) Designs are infinite objects in general, while effective computation requires of finitary representation. We therefore introduce a *generator* producing a design. In particular, finite generators, which are analogous to automata, allow for finitary representation of some infinite designs.

(3) Designs in [9] capture only cut-free and identity-free proofs. While it is semantically natural (as strategies in game semantics are cut- and identity-free), it limits the computational power considerably. Hence we extend designs with explicit cuts (and also identities for future purposes).

We then study the basic properties of our extended designs and behaviours. Although most of them are adapted from the original work, our exposition puts special emphasis on their relevance to concrete computation.

It should be stressed that our purpose is not to replace the original framework, which has a lot of theoretical advantages, but to complement it with a handy extended syntax, which has practical advantages and is more oriented to applications.

The rest of this article is organized as follows. In Section 2, we introduce our syntax for designs, which simplifies and extends Curien's concrete syntax [2]. Our designs also incorporate explicit cuts and identities for computational purposes, and thus called *computational designs* or *c-designs*. *Design generators* producing c-designs are also introduced, which allow finite generation of infinite c-designs. They come equipped with a Krivine-style normalization procedure [15], that leads to effective computation over infinite c-designs. There is a quite satisfactory definition of data as c-designs in our framework. Based on them, some examples of computation are illustrated. In particular, we give a bidirectional correspondence between deterministic finite automata (DFA) and finitely generated *standard* designs, which are cut-free as in Girard's original definition. This way we estimate the computational power of finitely generated standard designs as that of DFA.

In Section 3, we study the analytical properties of designs from a computational point of view. *Associativity* of normalization is important for composition of function designs, while *separation* is for acceptance of data designs. The *pull-back* property, a ludics analogue of linearity, is useful for acceptance of sets of data designs. In passing, we also observe that the computational power of finitely generated c-designs goes far more beyond DFAs, once equipped with cuts; indeed they capture all recursively enumerable languages. This comes in contrast with the cut-free case above, and in fact was our original motivation to consider designs with cuts.

In Section 4, we introduce the *behaviours*, i.e. biorthogonal-closed sets of (linear) c-designs. Behaviours may be considered as generalizations of languages. To have an exact correspondence, however, one has to restrict a behaviour to the set of "pure" elements in it. Here the notion of *incarnation*, a truly original discovery of ludics, plays an essential role. While interactive definition of languages via machines/automata is well expressed by orthogonal construction of behaviours, constructive definition of languages via language operators (e.g., union and Kleene's star) is supported by logical construction based on *logical connectives*. Here, *internal completeness*, another originality of ludics, plays a key role. We end the section by exhibiting ludics analogues of some language operators.

Section 5 concludes the article with a number of future research directions.

2 Designs and Normalization

We introduce our new notion of design, which modifies Girard's original one (subsection 2.1). They can be generated by design generators, sometimes by finite ones (subsection 2.2). Normalization of designs is defined in two ways, first by a reduction-based procedure (subsection 2.3) and later by a Krivine-style one (subsection 2.6). We also illustrate how to represent data and functions as designs (subsection 2.4), and show that finitely generated cut-free designs correspond to deterministic finite automata, and thus have a limited computational power (subsection 2.5).

2.1 Designs

We present a handy syntax for designs that simplifies and extends Curien's concrete syntax [2]. Inspired by a close relationship with linear π-calculus [7], we adopt a notation analogous to π-calculus.

As with Curien's, it can be best understood by analogy with lambda terms. Let us consider the simple types generated by $\tau ::= \iota \mid \tau \to \tau$ and a fragment of simply typed lambda calculus given by:

$$\begin{aligned} P^{\iota} &::= (N_0^{\tau_1 \to \dots \tau_n \to \iota}) N_1^{\tau_1} \dots N_n^{\tau_n}, \\ N^{\tau_1 \to \dots \tau_n \to \iota} &::= x \mid \lambda x_1^{\tau_1} \cdots x_n^{\tau_n} . P^{\iota}. \end{aligned}$$

The terms of the form P (resp. N) are considered *positive* (resp. *negative*).

Positive terms are always of atomic type, and take some number of arguments, while negative ones are of arbitrary type, and among them non-variable ones bind some number of variables. A redex is a positive term of the form $(\lambda x_1 \cdots x_n.P)N_1 \cdots N_n$. Because of the typing, the *arity* n always agrees. Hence one can apply n steps of reduction at once:

$$(\lambda x_1 \cdots x_n.P)N_1 \cdots N_n \longrightarrow P[N_1/x_1, \ldots, N_n/x_n]$$

yielding another positive term. Two restrictions may be imposed. A term is *normal* if in any positive subterm $(N_0)N_1 \ldots N_n$, N_0 is a variable. On the other hand, a term is *η-long* if in any positive subterm $(N_0)N_1 \ldots N_n$, none of $N_1, \ldots, N_n$ is a variable, unless it is of atomic type.

The designs of ludics extend the lambda terms in this well-behaved fragment in several ways. First, designs can be infinitary. Second, types are dropped and agreement of arity is ensured in another way. Third, instead of the single constructor/destructor pair, that is λ and the application, there are plenty of such pairs, one for each finite set I of natural numbers (called a *ramification*). A special term for termination (called the *daimon*) is also added, and finally additive superimposition $N_1 + N_2 + \cdots$ of terms is allowed.

Actually, the original designs extend the normal, η-long and linear lambda terms. In contrast, our syntax also encompasses non-normal, non-η-long and non-linear terms. Another difference is that terms are built from an arbitrary set of names, rather than the fixed set of ramifications.

Definition 2.1 A *signature* $\mathcal{A}$ is a pair (A, ar) of a set A of *names* and a function $ar : A \longrightarrow \mathbb{N}$ giving an *arity* to each name.

Let $\mathcal{V}$ be a denumerable set of variables $x, y, z, \ldots$. We build *actions* from a given signature $\mathcal{A}$ and $\mathcal{V}$. A *positive action* is either ✠ (*daimon*), Ω (*divergence*) or $\overline{a}$ with $a \in A$ (*proper positive action*). A *negative action* is either $x \in \mathcal{V}$ (*variable*) or $a(x_1, \ldots, x_n)$ (*proper negative action*) where $a \in A$, $ar(a) = n$ and $x_1, \ldots, x_n$ are distinct variables. In the sequel, we adopt the following convention: each of $\vec{x}_a, \vec{y}_a, \ldots$ denotes a vector of $n = ar(a)$ distinct variables. Hence an expression of the form $a(\vec{x}_a)$ always denotes a negative action.

Remark 2.2 Names generalize ramifications $I \in \mathcal{P}_f(\mathbb{N})$ of the original ludics. In fact, the original designs can be considered as structures over the signature $\mathcal{RAM} = (\mathcal{P}_f(\mathbb{N}), |\ \ |)$, where $|I|$ gives the cardinality of $I \in \mathcal{P}_f(\mathbb{N})$. Our use of names allows for a handy notation and circumvents the difficulty associated to the empty ramification (see 5.2.4 of [9]).

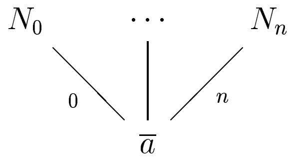
We are now ready to define our version of designs. To distinguish them from the original ones, we call them *computational designs*, or *c-designs*.

Definition 2.3 We fix a signature $\mathcal{A} = (A, ar)$. Let $\mathcal{T}$ be the set of (possibly infinite) rooted trees in which each node is labelled with a positive action, a variable, or a set $\{a(\vec{x}_a)\}_{a \in A}$ of proper negative actions indexed by A, and each edge is labelled with $l \in \mathbb{N} \cup A$.

The set $\mathcal{D}^+$ of *positive c-designs* and the set $\mathcal{D}^-$ of *negative c-designs* are the largest subsets of $\mathcal{T}$ that satisfy the following conditions.

1. If $P \in \mathcal{D}^+$, then one of the following holds:

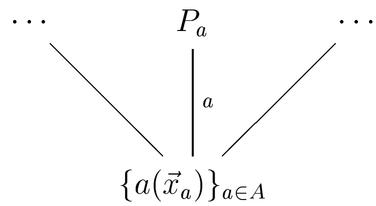
- P is a single node labelled with $\maltese$.
- P is a single node labelled with Ω.
- P is of the form



  i.e. a tree whose root is labelled with a positive action $\overline{a}$ with $ar(a) = n$ and has $n+1$ immediate subtrees $N_0, \ldots, N_n \in \mathcal{D}^-$. The edge connecting the root to N_i is labelled with $i \in \mathbb{N}$. We denote P by $N_0 | \overline{a}\langle N_1, \ldots, N_n \rangle$.

2. If $N \in \mathcal{D}^-$, then one of the following holds:

- N is a single node labelled with a variable x.
- N is of the form



  i.e. a tree whose root is labelled with $\{a(\vec{x}_a)\}_{a \in A}$ and has $|A|$ immediate subtrees $\{P_a\}_{a \in A}$, all in $\mathcal{D}^+$. The edge connecting the root to P_a is labelled with $a \in A$. We denote N by $\sum a(\vec{x}_a).P_a$.

Informally, we may consider $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$ to be coinductively defined by

$$P ::= \maltese \mid \Omega \mid N_0 | \overline{a}\langle N_1, \ldots, N_n \rangle,$$
$$N ::= x \mid \sum a(\vec{x}_a).P_a.$$

We use symbols $P, Q, \ldots$ for a positive c-design in $\mathcal{D}^+$, $M, N, \ldots$ for a negative c-design in $\mathcal{D}^-$, and $T, U, \ldots$ for an arbitrary one in $\mathcal{D}$.

By definition, every non-variable negative c-design is fully branching, i.e., has $|A|$-many children, that is often too much. A partially branching one can be encoded by using Ω. Given a subset $K \subseteq A$ and $\{P_a\}_{a\in K}$, we write $\sum_K a(\vec{x}_a).P_a$ to denote the negative c-design $\sum a(\vec{x}_a).Q_a$ where $Q_a = P_a$ if $a \in K$ and $Q_a = \Omega$ otherwise. When K is a finite set $\{a_1, \ldots, a_n\}$, we use the notation $a_1(\vec{x}_1).P_{a_1} + \cdots + a_n(\vec{x}_n).P_{a_n}$. In particular, when K is a singleton $\{a\}$ or the empty set, we write $a(\vec{x}_a).P_a$ or 0, respectively.

A positive c-design $N_0|\overline{a}\langle\rangle$ with 0-ary name a is simply written as $N_0|\overline{a}$.

A subtree of T is called a *subdesign* of T.

Definition 2.4

- A positive c-design of the form $N_0|\overline{a}\langle N_1, \ldots, N_n\rangle$ is called a *cut* if N_0 is not a variable, and hence is of the form $(\sum a(\vec{x}_a).P_a)\,|\overline{a}\langle N_1, \ldots, N_n\rangle$.
- A variable x occurring as $N_0|\overline{a}\langle N_1, \ldots, x, \ldots, N_n\rangle$ in T is called an *identity* in T. If $T = x$, then T itself is an identity.

We call T *cut-free* (*identity-free*, resp.) if it does not contain a cut (identity, resp.) as subdesign.

If T is cut- and identity-free, any positive subterm is either $\maltese$, Ω or of the form $x|\overline{a}\langle N_1, \ldots, N_n\rangle$ where none of $N_1, \ldots, N_n$ is a variable. Observe the analogy with the normal and η-long terms in the well-behaved fragment of lambda calculus given at the beginning of this subsection. Furthermore, anticipating subsection 2.3, the reduction rule for c-designs is as follows:

$$(\textstyle\sum a(x_1, \ldots, x_n).P_a)\,|\overline{a}\langle N_1, \ldots, N_n\rangle \longrightarrow P_a[N_1/x_1, \ldots, N_n/x_n].$$

This is also analogous to the reduction rule for lambda terms:

$$(\lambda x_1 \cdots x_n.P)N_1 \cdots N_n \longrightarrow P[N_1/x_1, \ldots, N_n/x_n].$$

Our c-designs involve binding expressions $a(\vec{x}_a).P_a$ which binds free occurrences of $\vec{x}_a$ in P_a. Hence it is natural to identify them up to α-equivalence. By *renaming* we mean a function $\rho : \mathcal{V} \longrightarrow \mathcal{V}$. We write *id* for the identity renaming, and $\rho[z/x]$ for the renaming that agrees with ρ except that $\rho[z/x](x) = z$. The set of renamings is denoted by $\mathcal{RN}$.

Definition 2.5 The *α-equivalence* is the largest relation $R \subseteq (\mathcal{D} \times \mathcal{RN})^2$ such that if $(T, \rho)\ R\ (U, \tau)$, then one of the following holds:

(1) $T = \maltese = U$;
(2) $T = \Omega = U$;

(3) $T = N_0|\overline{a}\langle N_1, \ldots, N_n\rangle$, $U = M_0|\overline{a}\langle M_1, \ldots, M_n\rangle$ and $(N_k, \rho)\ R\ (M_k, \tau)$ for every $0 \leq k \leq n$;
(4) $T = x$, $U = y$ and $\rho(x) = \tau(y)$;
(5) $T = \sum a(\vec{x}_a).P_a$, $U = \sum a(\vec{y}_a).Q_a$ and $(P_a, \rho[\vec{z}_a/\vec{x}_a])\ R\ (Q_a, \rho[\vec{z}_a/\vec{y}_a])$ for every $a \in A$ and some vector $\vec{z}_a$ of fresh variables.

T and U are *α-equivalent* if $(T, id)\ R\ (U, id)$.

In the sequel, we identify c-designs up to α-equivalence.

Given a c-design T, the set of free variables in it is denoted by $\mathsf{fv}(T)$. We omit a formal definition, as it is intuitively clear.

If T is a c-design and N a negative c-design, $T[N/x]$ denotes the c-design obtained from T by substituting N for all free occurrences of x in T. In doing so, we assume that the bound variables of T have been suitably renamed, so that no free variable of N is newly bound by the substitution.

The following lemma is useful when proving two c-designs are equivalent (up to α-equivalence).

Lemma 2.6 *Let R be a binary relation on c-designs such that*

- *R is closed under α-equivalence: if T and T' (resp. U and U') are α-equivalent and $T\ R\ U$, then $T'\ R\ U'$;*
- *if $T\ R\ U$ then one of the following holds:*

(1) $T = \maltese = U$;
(2) $T = \Omega = U$;
(3) $T = N_0|\overline{a}\langle N_1, \ldots, N_n\rangle$, $U = M_0|\overline{a}\langle M_1, \ldots, M_n\rangle$ and $N_k\ R\ M_k$ for every $0 \leq k \leq n$;
(4) $T = x = U$;
(5) $T = \sum a(\vec{x}_a).P_a$, $U = \sum a(\vec{x}_a).Q_a$ and $P_a\ R\ Q_a$ for every $a \in A$.

If $T\ R\ U$, then T and U are α-equivalent.

Proof Define a new relation $R' \subseteq (\mathcal{D} \times \mathcal{RN})^2$ as follows:

- $(T, \rho)\ R'\ (U, \tau)$ if $T\rho\ R\ U\tau$, where $T\rho$ is the result of applying the renaming ρ to the free occurrences of variables in T.

Assume that $(T, \rho)\ R'\ (U, \tau)$ and verify that one of (1) – (5) in Definition 2.5 holds for R'. The most crucial case is when T is of the form $\sum a(\vec{x}_a).P_a$ so that $T\rho = (\sum a(\vec{x}_a).P_a)\,\rho = \sum a(\vec{x}_a).(P_a\rho[\vec{x}_a/\vec{x}_a])$. Since $T\rho\ R\ U\tau$, U must be of the form $\sum a(\vec{y}_a).Q_a$ so that $U\tau = \sum a(\vec{y}_a).(Q_a\tau[\vec{y}_a/\vec{y}_a])$. Let $a \in A$ and $\vec{z}_a$ be a vector of fresh variables. Since R is closed under α-equivalence, we have $\sum a(\vec{z}_a).(P_a\rho[\vec{z}_a/\vec{x}_a])\ R\ \sum a(\vec{z}_a).(Q_a\tau[\vec{z}_a/\vec{y}_a])$, and so $P_a\rho[\vec{z}_a/\vec{x}_a]\ R\ Q_a\tau[\vec{z}_a/\vec{y}_a]$

by (5). This proves that $(P_a, \rho[\vec{z}_a/\vec{x}_a])\ R'\ (Q_a, \tau[\vec{z}_a/\vec{y}_a])$ for every $a \in A$, as required. $\square$

Definition 2.7 Let T be a c-design.

- T is *total* if $T \neq \Omega$.
- T is *linear* if for any subdesign of the form $N_0|\overline{a}\langle N_1, \dots, N_n\rangle$, the sets $\mathsf{fv}(N_0)$, $\dots$, $\mathsf{fv}(N_n)$ are pairwise disjoint.
- T is *standard* if it is total, linear, cut-free, identity-free and $\mathsf{fv}(T)$ is finite.

Remark 2.8 The standard c-designs over the signature $\mathcal{RAM}$ exactly correspond to the original designs in [9].

Concretely, let T be a standard c-design with free variables $\{x_1, \dots, x_n\}$. Assuming that a distinct address ξ_i (locus) is associated to each x_i, and also assuming that an address ξ_0 is associated to the root when T is negative, one can automatically infer the addresses of its subdesigns. Let $P = y|\overline{J}\langle\vec{M}\rangle$ be an occurrence of a positive subdesign of T. If $y = x_i$, then its address is ξ_i. Otherwise, the address is obtained via the address ξ of the binder $\sum I(\vec{y}_I).Q_I$ of y. If $I = \{i_1, \dots, i_m\}$ with $i_1 < \dots < i_m$, $\vec{y}_I = y_{i_1}, \dots, y_{i_m}$, and $y = y_{i_k}$, then the address of P is $\xi.i_k$. The address of an occurrence of a negative subdesign is obtained from the positive one immediately above it. Let $P = y|\overline{I}\langle N_1, \dots, N_m\rangle$ be a subdesign of T, where $I = \{i_1, \dots, i_m\}$, $i_1 < \dots < i_m$. If the address of P is ξ, the address of N_k is $\xi.i_k$ for each $1 \leq k \leq m$.

2.2 Design generators

Since designs are infinitary in general, they are not directly an object of effective computation. We therefore introduce design generators that provide a means to finitely describe infinite designs. Generators are also useful for defining a function on designs by corecursion.

Definition 2.9 A *generator* G is a triple (S^+, S^-, ℓ) where S^+ and S^- are disjoint sets of *states*, and ℓ is a function defined on $S = S^+ \cup S^-$ which satisfies the following conditions:

- For $s^+ \in S^+$, $\ell(s^+)$ is either ✠, Ω or an expression of the form $s_0^-|\overline{a}\langle s_1^-, \dots, s_n^-\rangle$ such that $a \in A$, $ar(a) = n$ and $s_0^-, \dots, s_n^- \in S^-$.
- For $s^- \in S^-$, $\ell(s^-)$ is either a variable x or an expression of the form $\sum a(\vec{x}_a).s_a^+$ such that $s_a^+ \in S^+$ for every $a \in A$.

A *pointed* generator is a pair (G, s_I) of a generator $G = (S^+, S^-, \ell)$ and $s_I \in S$. It is also written as a quadruple (S^+, S^-, ℓ, s_I).

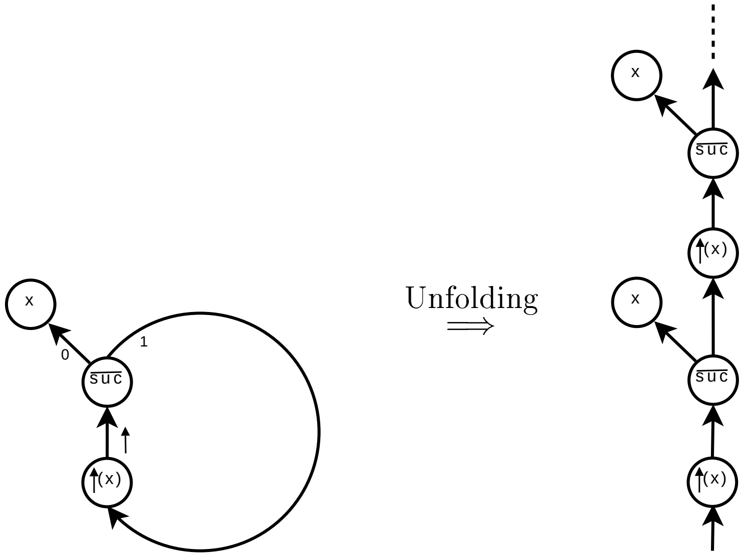


Fig. 1. Generator for $\omega^\star$ and its unfolding

A generator G can be considered as a labelled directed graph with the set S of vertices. The equation $\ell(s^+) = s_0^- |\overline{a}\langle s_1^-, \ldots, s_n^-\rangle$ can be read as "the vertex s^+ has label $\overline{a}$ and there is a labelled edge $s^+ \xrightarrow{i} s_i^-$ for every $0 \leq i \leq n$." Likewise, $\ell(s^-) = \sum a(\vec{x}_a).s_a^+$ can be read as "the vertex s^- has label $\{a(\vec{x}_a)\}_{a\in A}$ and there is a labelled edge $s^- \xrightarrow{a} s_a^+$ for every $a \in A$." Hence given an initial point s_I, the standard unfolding procedure yields a a labelled tree with root s_I. It is in fact a c-design, which we denote by $\mathsf{design}(G, s_I)$. We say that (G, s_I) *generates* the c-design $\mathsf{design}(G, s_I)$.

For instance, the pointed generator $(\{s_{\maltese}\}, \{s_N\}, \ell, s_N)$ with

$$\ell(s_{\maltese}) = \maltese, \qquad \ell(s_N) = \sum a(\vec{x}_a).s_{\maltese}$$

generates the *negative daimon* $\maltese^- = \sum a(\vec{x}).\maltese$. On the other hand, consider $(\{s_P, s_\Omega\}, \{s_N, s_x\}, \ell, s_N)$ with

$$\ell(s_P) = s_x|\overline{\mathsf{suc}}\langle s_N\rangle, \qquad \ell(s_N) = \uparrow(x).s_P, \qquad \ell(s_x) = x, \qquad \ell(s_\Omega) = \Omega,$$

Here, $\uparrow(x).s_P$ is a shorthand for $\sum a(\vec{x}).s_a$ where s_a is s_P if $a = \uparrow$ and is s_Ω otherwise (see Fig. 1). It generates an infinite negative c-design $\omega^\star = \uparrow(x).x|\overline{\mathsf{suc}}\langle\omega^\star\rangle$ that will be considered as a representation of the ordinal ω (see subsection 2.4).

There is a *universal generator* $G_{unv} = (\mathcal{D}^+, \mathcal{D}^-, id)$, which consists of the set of all positive c-designs, that of all negative c-designs, and the identity function. Notice that the identity function id on $\mathcal{D}^+ \cup \mathcal{D}^-$ can be (abusively) considered as a labelling function ℓ in the sense of Definition 2.9. Hence every c-design U is generated by a pointed generator. In fact, we have $\mathsf{design}(G_{unv}, U) = U$.

Different generators may generate the same c-design. A condition for equivalence can be given by bisimulation, though we do not pursue it in the current article.

Definition 2.10 A c-design U is *finitely generated* if it is generated by a pointed generator (G, s_I) which has finitely many states, and whenever $\ell(s^-) = \sum a(\vec{x}_a).s_a$, all but finitely many s_a have the label Ω.

For instance, the above $\omega^\star$ is finitely generated. The power of finite generation is partly witnessed by the following proposition; notice that x may occur infinitely many times in T below.

Proposition 2.11 *If T and N are finitely generated, so is $T[N/x]$.*

By using generators, we can easily justify corecursive definition of functions on c-designs.

Theorem 2.12 *Let $f : \mathcal{D} \longrightarrow \mathcal{D}$ be a function that respects the polarity (i.e. maps a positive c-design to a positive one, etc.). Then there exists a unique function $\hat{f} : \mathcal{D} \longrightarrow \mathcal{D}$ such that*

$$\begin{aligned}
\hat{f}(P) &= \hat{f}(N_0)|\overline{a}\langle \hat{f}(N_1), \dots, \hat{f}(N_n)\rangle && \textit{if } f(P) = N_0|\overline{a}\langle N_1, \dots, N_n\rangle; \\
&= \maltese && \textit{if } f(P) = \maltese; \\
&= \Omega && \textit{if } f(P) = \Omega; \\
\hat{f}(N) &= \textstyle\sum a(\vec{x}_a).\hat{f}(P_a) && \textit{if } f(N) = \textstyle\sum a(\vec{x}_a).P_a; \\
&= x && \textit{if } f(N) = x.
\end{aligned}$$

Proof Let G_f be the generator $(\mathcal{D}^+, \mathcal{D}^-, f)$, and define $\hat{f}(U) = \mathsf{design}(G_f, U)$ (here we abusively think of f as the labelling function ℓ as in the definition of the universal generator). Then it is easy to verify the above equations. For instance, if $f(P) = N_0|\overline{a}\langle N_1, \dots, N_n\rangle$, then

$$\begin{aligned}
\hat{f}(P) &= \mathsf{design}(G_f, P) \\
&= \mathsf{design}(G_f, N_0)|\overline{a}\langle \mathsf{design}(G_f, N_1), \dots, \mathsf{design}(G_f, N_n)\rangle \\
&= \hat{f}(N_0)|\overline{a}\langle \hat{f}(N_1), \dots, \hat{f}(N_n)\rangle.
\end{aligned}$$

The uniqueness can be established by a standard bisimulation argument. $\square$

Designs can be normalized in several ways. We first present a reduction-based procedure. It is defined in two stages. First, we introduce a reduction rule that finds a 'head normal form' whenever it exists.

Definition 2.13 The reduction relation $\longrightarrow$ is defined on positive c-designs by:

$$(\sum a(\vec{x}_a).P_a)\,|\overline{a}\langle\vec{N}\rangle \longrightarrow P_a[\vec{N}/\vec{x}_a],$$

where $\vec{N}$ is a vector of $n = ar(a)$ negative c-designs. The transitive reflexive closure of $\longrightarrow$ is denoted by by $\longrightarrow^*$. We write $P \Downarrow Q$ if $P \longrightarrow^* Q$ and Q is neither a cut nor Ω. If there is no such Q, we write $P \Uparrow$.

When P is closed (i.e. has no free variables), P is a cut, ✠ or Ω. Hence we have either $P \Downarrow$ ✠ or $P \Uparrow$.

Second, we expand $\longrightarrow$ by corecursion. Define a function $hnf : \mathcal{D} \longrightarrow \mathcal{D}$ by $hnf(P) = Q$ if $P \Downarrow Q$ and $hnf(P) = \Omega$ otherwise; hnf is just the identity on negative c-designs. Then Theorem 2.12 ensures the unique existence of the following function.

Definition 2.14 The *normal form function* $[\![\quad]\!] : \mathcal{D} \longrightarrow \mathcal{D}$ is defined as follows:

$$\begin{aligned}
[\![P]\!] &= \text{✠} && \text{if } P \Downarrow \text{✠};\\
&= \Omega && \text{if } P \Uparrow;\\
&= x|\overline{a}\langle[\![N_1]\!],\ldots,[\![N_n]\!]\rangle && \text{if } P \Downarrow x|\overline{a}\langle N_1,\ldots,N_n\rangle;\\
[\![x]\!] &= x;\\
[\![\sum a(\vec{x}_a).P_a]\!] &= \sum a(\vec{x}_a).[\![P_a]\!].
\end{aligned}$$

Though intuitive, the above definition of normalization is not effective, because it works on c-designs which are often infinite, and the substitution involves a costly renaming operation. An alternative procedure will be given in subsection 2.6.

To give an example of normalization, let us consider the *fax*, that is a fully η-expanded form of the identity axiom. It is defined by a recursive equation:

$$\eta(N) = \sum a(y_1,\ldots,y_n).N|\overline{a}\langle\eta(y_1),\ldots,\eta(y_n)\rangle,$$

where n varies depending on the arity of each name $a \in A$. It is easy to give a finite generator for it, whenever A is finite. The fax $\eta(x)$ is standard, and it works as the identity function when applied to cut- and identity-free c-designs:

Proposition 2.15 *Let P and N be respectively positive and negative c-designs without cuts and identities. For any variables $x_1, \ldots, x_n$, we have*

$$[\![P[\eta(x_1)/x_1, \ldots, \eta(x_n)/x_n]]\!] = P, \qquad [\![\eta(N[\eta(x_1)/x_1, \ldots, \eta(x_n)/x_n])]\!] = N.$$

Proof Define a binary relation R on $\mathcal{D}$ as follows:

- $P \; R \; Q$ if P is cut- and identity-free, and $Q = [\![P[\eta(x_1)/x_1, \ldots, \eta(x_n)/x_n]]\!]$ for some $x_1, \ldots, x_n$;
- $N \; R \; M$ if N is cut- and identity-free (and hence is not a variable), and $M = [\![\eta(N[\eta(x_1)/x_1, \ldots, \eta(x_n)/x_n])]\!]$ for some $x_1, \ldots, x_n$;
- $x \; R \; x$ for any variable x.

Let us verify that if $T \; R \; U$, one of (1) – (5) in Lemma 2.6 holds.

If T is of the form $x|\overline{a}\langle N_1, \ldots, N_m\rangle$, then $U = [\![T[\eta(x_1)/x_1, \ldots, \eta(x_n)/x_n]]\!]$. Assume $x = x_i$ for some $1 \le i \le n$ (otherwise the proof is easier). We write $[\eta(\vec{x})/\vec{x}]$ for $[\eta(x_1)/x_1, \ldots, \eta(x_n)/x_n]$. Then,

$$\begin{aligned}
U &= [\![\eta(x)|\overline{a}\langle N_1[\eta(\vec{x})/\vec{x}], \ldots, N_m[\eta(\vec{x})/\vec{x}]\rangle]\!] \\
&= [\![x|\overline{a}\langle \eta(N_1[\eta(\vec{x})/\vec{x}]), \ldots, \eta(N_m[\eta(\vec{x})/\vec{x}])\rangle]\!] \\
&= x|\overline{a}\langle [\![\eta(N_1[\eta(\vec{x})/\vec{x}])]\!], \ldots, [\![\eta(N_m[\eta(\vec{x})/\vec{x}])]\!]\rangle.
\end{aligned}$$

Since $x \; R \; x$ and $N_i \; R \; [\![\eta(N_i[\eta(\vec{x})/\vec{x}])]\!]$ for every $1 \le i \le m$, (3) holds.

When T is of the form $\sum a(\vec{z}_a).P_a$, we may assume that $x_1, \ldots, x_n$ are distinct from $\vec{z}_a$ by α-equivalence. Now,

$$\begin{aligned}
U &= [\![\eta(T[\eta(\vec{x})/\vec{x}])]\!] \\
&= [\![\textstyle\sum a(y_1, \ldots, y_n).\left(\sum a(\vec{z}_a).P_a[\eta(\vec{x})/\vec{x}]\right) |\overline{a}\langle \eta(y_1), \ldots, \eta(y_n)\rangle]\!] \\
&= \textstyle\sum a(y_1, \ldots, y_n).[\![\left(\sum a(\vec{z}_a).P_a[\eta(\vec{x})/\vec{x}]\right) |\overline{a}\langle \eta(y_1), \ldots, \eta(y_n)\rangle]\!] \\
&= \textstyle\sum a(y_1, \ldots, y_n).[\![P_a[\eta(\vec{x})/\vec{x}, \eta(y_1)/z_1, \ldots, \eta(y_n)/z_n]]\!] \\
&= \textstyle\sum a(z_1, \ldots, z_n).[\![P_a[\eta(\vec{x})/\vec{x}, \eta(z_1)/z_1, \ldots, \eta(z_n)/z_n]]\!].
\end{aligned}$$

Since $P_a \; R \; [\![P_a[\eta(\vec{x})/\vec{x}, \eta(z_1)/z_1, \ldots, \eta(z_n)/z_n]]\!]$, (5) holds.

Other cases are straightforward. Hence by Lemma 2.6, we obtain the desired equalities. $\square$

A consequence is that one can safely replace an identity x with $\eta(x)$ in some situations (see Remark 2.18).

Let us now discuss representations of data and functions in ludics. First of all, note that if a signature $\mathcal{A}$ contains a 0-ary name n for each natural number $n \in \mathbb{N}$, an arbitrary function $f : \mathbb{N} \longrightarrow \mathbb{N}$ can be represented by a c-design $\sum_{\mathbb{N}} n.\overline{f(n)}$. In fact, we have

$$\left(\textstyle\sum_{\mathbb{N}} n.\overline{f(n)}\right) |\overline{m} \longrightarrow \overline{f(m)}.$$

But this does not admit a finite generator, hence is not interesting from a computational point of view. We are rather interested in structured data and finitely presentable functions over them.

Fortunately, ludics admits a quite general definition of data that encompasses most of what are usually called (first order) data. Throughout the rest of this paper, we assume that the signature $\mathcal{A}$ contains a fixed unary name $\uparrow$. We denote $\overline{\uparrow}$ by $\downarrow$. A negative c-design of the form $\uparrow(x).x|\overline{a}\langle\vec{N}\rangle$ with $x \notin \mathsf{fv}(\vec{N})$ is shorthanded by $\uparrow\overline{a}\langle\vec{N}\rangle$. It behaves as follows:

$$\uparrow\overline{a}\langle\vec{N}\rangle| \downarrow\langle M\rangle \longrightarrow M|\overline{a}\langle\vec{N}\rangle.$$

Definition 2.16 The set of *data designs* consists of negative c-designs $d, d_1, d_2, \ldots$ coinductively defined as follows:

$$d \ ::= \uparrow\overline{a}\langle d_1, \ldots, d_n\rangle,$$

where a stands for an arbitrary name and $n = ar(a)$.

Data designs are standard, and the only negative action involved is $\uparrow(x)$. Furthermore, the variable x thus introduced is immediately consumed. Hence the binding relation, which corresponds to the justification relation in Hyland-Ong game semantics [11], is trivial.

For the purpose of giving examples, let Σ be an alphabet (i.e., a finite set of symbols) and consider a signature $\mathcal{A}_0$ that contains:

- 0-ary names: $\mathsf{zero}, \mathsf{nil}$, a^0 for each $a \in \Sigma$;
- unary names: suc, a^1 for each $a \in \Sigma$;
- binary names: $\mathsf{pair}, \mathsf{cons}$, a^2 for each $a \in \Sigma$.

In the sequel, a^0, a^1 and a^2 are often written as a.

Each natural number n can be represented by a data design $n^\star$:

$$\begin{aligned} 0^\star &= \uparrow\overline{\mathsf{zero}} &&= \uparrow(x).x|\overline{\mathsf{zero}} \\ (n+1)^\star &= \uparrow\overline{\mathsf{suc}}\langle n^\star\rangle &&= \uparrow(x).x|\overline{\mathsf{suc}}\langle n^\star\rangle \end{aligned}$$

A data design $\omega^\star$ corresponding to the ordinal ω can also be defined by a recursive equation:

$$\omega^\star \ = \uparrow\overline{\mathsf{suc}}\langle\omega^\star\rangle.$$

As we have seen in subsection 2.2, $\omega^\star$ is finitely generated.

Similarly, words over Σ, labelled binary trees over Σ, and lists over a set $\mathbf{D}$ of data designs are represented as follows:

$$\begin{array}{ccc} \epsilon^\star = \uparrow\overline{\mathsf{nil}}, & \mathtt{leaf}_a^\star = \uparrow\overline{a}^0, & []^\star = \uparrow\overline{\mathsf{nil}}; \\ (aw)^\star = \uparrow\overline{a}^1\langle w^\star\rangle, & (\mathtt{node}_a(t,u))^\star = \uparrow\overline{a}^2\langle t^\star, u^\star\rangle, & (d :: l)^\star = \uparrow\overline{\mathsf{cons}}\langle d, l^\star\rangle. \end{array}$$

where $a \in \Sigma$ and $w \in \Sigma^*$. $\mathtt{leaf}_a$ is a single node labelled with a, $\mathtt{node}_a(t,u)$ is a tree with root labelled by a and has immediate subtrees t, u. $[]$ is the empty list, $d \in \mathbf{D}$, and l stands for a list over $\mathbf{D}$. These representations can be extended to the infinitary ones in the same way as natural numbers are extended to ω.

We have chosen data to be negative c-designs, even though they are positive "in spirit," as their main ingredients are positive actions (the negative action $\uparrow$ is just used for adjusting polarity). The reason is that a c-design may in general have multiple variables, for which *negative* c-designs can be substituted. Hence our choice allows for natural definitions of multi-arity functions.

Definition 2.17 An *n-ary function design* is a negative c-design $F[x_1,\ldots,x_n]$ such that $\mathsf{fv}(F) \subseteq \{x_1,\ldots,x_n\}$ and $[\![F[d_1,\ldots,d_n]]\!]$ is either a data design or $\uparrow(x).\Omega$ for any data $d_1,\ldots,d_n$.

Notice that $\uparrow(x).\Omega$ can also be written as $\sum a(\vec{x}_a).\Omega$. It is a negative version of divergence Ω, and is called *skunk* in [9]. In the sequel, we give some typical examples of function designs.

Constructors. To each $a \in A$ of arity n, an n-ary function design $\uparrow\overline{a}\langle\vec{x}_a\rangle$ is associated, representing the constructor function for a. For instance, the successor for the natural numbers is given by $\mathsf{Suc}[x] =\uparrow\overline{\mathsf{suc}}\langle x\rangle$.

Discriminators. Let $K \subseteq A$ and suppose that a function design $F_a[\vec{x}_a]$ is given for each $a \in K$. We define

$$\mathsf{case}\ x\ \mathsf{of}\ \{a(\vec{x}_a) \Rightarrow F_a[\vec{x}_a]\}_{a\in K} = \uparrow(y).x|\downarrow\Big\langle \textstyle\sum_K a(\vec{x}_a).(F_a[\vec{x}_a]|\downarrow\langle y\rangle)\Big\rangle.$$

Given a data design $d = \uparrow\overline{a}\langle d_1, \ldots, d_n\rangle$ with $a \in K$, it works as follows (below, we write $T \Longrightarrow U$ if U is obtained by applying the reduction rule to a *subdesign* of T).

$$\begin{aligned}
\mathsf{case}\ d\ \mathsf{of}\ \{a(\vec{x}_a) \Rightarrow F_a[\vec{x}_a]\}_{a\in K} &= \uparrow(y).d|\downarrow\Big\langle \textstyle\sum_K a(\vec{x}_a).(F_a[\vec{x}_a]|\downarrow\langle y\rangle)\Big\rangle \\
&\Longrightarrow \uparrow(y).\Big(\textstyle\sum_K a(\vec{x}_a).(F_a[\vec{x}_a]|\downarrow\langle y\rangle)\Big)|\overline{a}\langle d_1, \ldots, d_n\rangle \\
&\Longrightarrow \uparrow(y).(F_a[d_1, \ldots, d_n]|\downarrow\langle y\rangle).
\end{aligned}$$

Since F_a is a function design, the normal form $[\![F_a[d_1, \ldots, d_n]]\!]$ is of the form $\uparrow(x).P$ for some P. Hence we have

$$(!)\quad \uparrow(y).(\uparrow(x).P|\downarrow\langle y\rangle) \Longrightarrow \uparrow(y).P[y/x] \;= \uparrow(x).P.$$

Assuming associativity of normalization (Theorem 3.1), we obtain

$$[\![\mathsf{case}\ d\ \mathsf{of}\ \{a(\vec{x}_a) \Rightarrow F_a[\vec{x}_a]\}_{a\in K}]\!] = [\![F_a[d_1, \ldots, d_n]]\!].$$

By using this construction, the predecessor for natural numbers can be defined:

$$\mathsf{Pred}[x] = \mathsf{case}\ x\ \mathsf{of}\ \{\mathsf{zero} \Rightarrow 0^\star,\ \mathsf{suc}(z) \Rightarrow z\}.$$

Remark 2.18 Although the definition of $\mathsf{case}\ x\ \mathsf{of}\ \{a(\vec{x}_a) \Rightarrow F_a[\vec{x}_a]\}_{a\in K}$ involves an identity y, one can replace y with the fax $\eta(y)$ to obtain the same result. In fact, nothing changes until (!) above, and then we have

$$(!')\quad \uparrow(y).\Big(\uparrow(x).P|\downarrow\langle\eta(y)\rangle\Big) \Longrightarrow \uparrow(y).P[\eta(y)/x] \;= \uparrow(x).P[\eta(x)/x].$$

Since P is cut- and identity-free, the last c-design is equivalent to $\uparrow(x).P$ by Proposition 2.15.

Duplicator. A remarkable feature of data designs is that any *finite* one can be duplicated by a *linear* c-design. The duplicator is recursively defined as follows:

$$\mathsf{dup}[x] = \mathsf{case}\ x\ \mathsf{of}\ \left\{ \begin{array}{l} a(\vec{x}_a) \Rightarrow \uparrow(w).\ \mathsf{dup}[x_1] | \downarrow \langle \mathsf{pair}(y_1, z_1). \\ \qquad\qquad\quad \mathsf{dup}[x_2] | \downarrow \langle \mathsf{pair}(y_2, z_2). \\ \qquad\qquad\quad \vdots \\ \qquad\qquad\quad \mathsf{dup}[x_n] | \downarrow \langle \mathsf{pair}(y_n, z_n). \\ \qquad\qquad\qquad w | \overline{\mathsf{pair}} \langle \uparrow \overline{a} \langle \vec{y}_a \rangle, \uparrow \overline{a} \langle \vec{z}_a \rangle \rangle \rangle \dots \rangle \end{array} \right\}_{a \in A}$$

where n depends on the arity of each $a \in A$, $\vec{x}_a = x_1, \dots, x_n$, $\vec{y}_a = y_1, \dots, y_n$ and $\vec{z}_a = z_1, \dots, z_n$.

It is linear, finitely generated, and contains cuts and identities. Identities can be removed as for the discriminators, while *cuts are essential*; if it is normalized, the normal form may not be finitely generated.

To see how it works, let d be a finite data design of the form $\uparrow \overline{a} \langle d_1, \dots, d_n \rangle$. Then, assuming $[\![\mathsf{dup}[d_i]]\!] = \uparrow \overline{\mathsf{pair}} \langle d_i, d_i \rangle$ for $i = 1, \dots, n$ and associativity of normalization, one can verify that $[\![\mathsf{dup}[d]]\!] = \uparrow \overline{\mathsf{pair}} \langle d, d \rangle$. Notice however that the argument goes by induction on the structure of d, and thus does not work for infinite data designs. In fact, the duplicator will diverge when applied to an infinite one.

The duplicator allows sharing of inputs: given $F[x_1, x_2]$, we define

$$(\mathsf{let}\ y = x_1, x_2\ \mathsf{in}\ F[x_1, x_2]) = \uparrow(z).\mathsf{dup}[y] | \downarrow \Big\langle \mathsf{pair}(x_1, x_2).(F[x_1, x_2] | \downarrow \langle z \rangle) \Big\rangle.$$

We then have

$$\begin{aligned} \mathsf{let}\ d = x_1, x_2\ \mathsf{in}\ F[x_1, x_2] &= \uparrow(z).\mathsf{dup}[d] | \downarrow \Big\langle \mathsf{pair}(x_1, x_2).(F[x_1, x_2] | \downarrow \langle z \rangle) \Big\rangle \\ &\Longrightarrow \uparrow(z).\ \uparrow \overline{\mathsf{pair}} \langle d, d \rangle | \downarrow \Big\langle \mathsf{pair}(x_1, x_2).(F[x_1, x_2] | \downarrow \langle z \rangle) \Big\rangle \\ &\Longrightarrow \uparrow(z).\Big(\mathsf{pair}(x_1, x_2).F[x_1, x_2] | \downarrow \langle z \rangle \Big) | \overline{\mathsf{pair}} \langle d, d \rangle \\ &\Longrightarrow \uparrow(z).\Big(F[d, d] | \downarrow \langle z \rangle \Big). \end{aligned}$$

Therefore,

$$[\![\mathsf{let}\ d = x_1, x_2\ \mathsf{in}\ F[x_1, x_2]]\!] = [\![F[d, d]]\!].$$

We end this subsection by showing that the general recursion scheme is *linearly* available.

Proposition 2.19 *Let F be an $m+1$-ary function design. Then there exists an m-ary function design $\hat{F}$ such that*

$$[\![\hat{F}[\vec{d}\,]\,]\!] = [\![F[\hat{F}[\vec{d}], \vec{d}\,]\,]\!]$$

for all finite data designs $\vec{d} = d_1, \ldots, d_m$.

If F *is linear (resp. finitely generated), so is* $\hat{F}$.

Proof For simplicity, we assume that $m = 1$. $\hat{F}$ is defined by a recursive equation:

$$\hat{F}[z] \ = \ (\mathsf{let}\ z = z_1, z_2\ \mathsf{in}\ F[\hat{F}[z_1], z_2]).$$

Suppose that F is linear and finitely generated. Then it is clear that the c-design

$$F'[X, z] \ = \ (\mathsf{let}\ z = z, z_2\ \mathsf{in}\ F[X, z_2])$$

is also linear and finitely generated. Notice the mixed use of the same variable z for free and bound occurrences; the LHS occurrence of z in $z = z, z_2$ is free while the RHS one is bound. Because of this mixed use, we obtain an α-equivalent of $\hat{F}$ by iteratively substituting $F'[X, z]$ for X. More precisely, let $G = (S^+, S^-, \ell, s_I)$ be a finite pointed generator for F'. Let s_X be the state with $\ell(s_X) = X$. Consider a new function ℓ' that agrees with ℓ except that $\ell'(s_X) = \ell(s_I)$. Then $\hat{G} = (S^+, S^-, \ell', s_I)$ generates the linear c-design $\hat{F}$. $\square$

Remark 2.20 It is essential to use cuts in duplicator, general recursion, and function designs in general for finite generation. One can of course eliminate cuts from a function design $F[x]$ by normalization, but then the result $[\![F[x]]\!]$ would correspond to the *extension* (denotation) of $F[x]$, which is hardly representable by finite means.

2.5 Standard c-designs and finite automata

We now discuss the computational power of standard c-designs. They are indeed very weak due to the absence of cuts. To formally estimate their strength, we give a bidirectional correspondence between standard c-designs and deterministic finite automata. Thus, standard c-designs are as powerful as finite automata, when computing over words.

Let us recall the definition of deterministic finite automata.

Definition 2.21 A *deterministic finite automaton* (DFA) M is a tuple $(\Sigma, Q, \delta, q_0, Q_F)$ where Σ is an alphabet, Q is a finite set (of states), $\delta : Q \times \Sigma \longrightarrow Q$ (the transition function), $q_0 \in Q$ (the initial state), $Q_F \subseteq Q$ (the final states). We write $q_1 \stackrel{a}{\longrightarrow} q_2$ if $\delta(q_1, a) = q_2$.

M *accepts* a word $w = a_1 \cdots a_n$ $(n \geq 0)$ if there is a sequence of transitions starting from q_0 such that

$$q_0 \stackrel{a_1}{\longrightarrow} q_1 \stackrel{a_2}{\longrightarrow} \cdots \stackrel{a_n}{\longrightarrow} q_f \in Q_f.$$

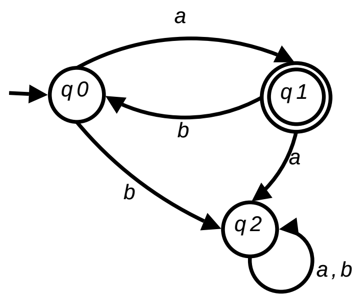


Fig. 2. DFA M_0

M *accepts* a language $\mathbf{L} \subseteq \Sigma^*$ if $\mathbf{L} = \{w \in \Sigma^* : \mathsf{M} \text{ accepts } w\}$. $\mathbf{L}$ is *regular* (*rational*) if $\mathbf{L}$ is accepted by a DFA.

We fix an alphabet Σ and work on a signature $\mathcal{A}$ which contains a unary name a for each $a \in \Sigma$. We can then associate to a language $\mathbf{L} \subseteq \Sigma^*$ a set $\mathbf{L}^\star = \{w^\star : w \in L\}$ of data designs.

Before stating the general result, let us give a simple example.

Example 2.22 Consider a DFA $\mathsf{M}_0 = (\{a, b\}, \{q_0, q_1, q_2\}, \delta, q_0, \{q_1\})$, where the transition relation δ is described in Fig. 2. M_0 accepts the language $a(ba)^*$.

This can be turned into a finitely generated standard c-design P_0 defined as follows:

$$\begin{aligned}
P_0 &= x|{\downarrow}\langle N_0\rangle, & N_0 &= a(x).P_1 + b(x).P_2 + \mathsf{nil}.\Omega,\\
P_1 &= x|{\downarrow}\langle N_1\rangle, & N_1 &= a(x).P_2 + b(x).P_0 + \mathsf{nil}.\maltese,\\
P_2 &= x|{\downarrow}\langle N_2\rangle, & N_2 &= a(x).P_2 + b(x).P_2 + \mathsf{nil}.\Omega.
\end{aligned}$$

Observe the correspondence between state q_i and c-design P_i for $i \in \{0, 1, 2\}$. In fact, we have $P_i[yw^\star/x] \longrightarrow^* P_j[w^\star/x]$ if and only if $q_i \xrightarrow{y} q_j$ for any $y \in \{a, b\}$ and $i, j \in \{0, 1, 2\}$. Hence $P_0[w^\star/x] \Downarrow \maltese$ if and only if $w \in a(ba)^*$.

Theorem 2.23 *For every DFA* M*, there exists a finitely generated positive standard c-design* P *such that*

(*) for any $w \in \Sigma^*$, M *accepts* w *if and only if* $P[w^\star/x] \Downarrow \maltese$.

Conversely, for every finitely generated positive standard c-design P *which has exactly one free variable* x*, there exists a DFA* M *such that (*) holds.*

Proof For simplicity, we assume that the alphabet Σ is $\{a, b\}$. We translate a given DFA $\mathsf{M} = (\Sigma, Q, \delta, q_0, Q_F)$ into a pointed generator (S^+, S^-, ℓ, s_I) as follows. When $Q = \{q_0, \dots, q_n\}$,

- $S^+ = \{q_0, \dots, q_n, q_\maltese, q_\Omega\}$; $S^- = \{s_0, \dots, s_n, s_x\}$; $s_I = q_0$;

- $\ell(q_i) = s_x | {\downarrow}\langle s_i \rangle$; $\ell(q_{\maltese}) = \maltese$; $\ell(q_\Omega) = \Omega$; $\ell(s_x) = x$;
- When $q_i \xrightarrow{a} q_j$ and $q_i \xrightarrow{b} q_k$,

$$\begin{aligned} \ell(s_i) &= a(x).q_j + b(x).q_k + \mathsf{nil}.q_{\maltese} \text{ if } q_i \in Q_F; \\ &= a(x).q_j + b(x).q_k + \mathsf{nil}.q_\Omega \text{ otherwise.} \end{aligned}$$

The generator is finite and generates positive standard c-designs $P_0, \ldots, P_n$ corresponding to $q_0, \ldots, q_n$ such that

$$P_i = x | {\downarrow}\langle a(x).P_j + b(x).P_k + \mathsf{nil}.R_i \rangle$$

when $q_i \xrightarrow{a} q_j$ and $q_i \xrightarrow{b} q_k$. R_i is $\maltese$ if $q_i \in Q_F$ and is Ω otherwise. It is easy to see that for any word w over $\{a, b\}$, $P_0[w^\star/x] \Downarrow \maltese$ iff M accepts w.

Conversely, given a finite pointed generator (G, s_I) with $G = (S^+, S^-, \ell)$ yielding a positive standard c-design P with exactly one free variable x, we build a finite automaton $\mathsf{M} = (\Sigma, Q, \delta, q_0, Q_F)$ as follows.

- $Q = S^+$; $q_0 = s_I$;
- For each $s \in S^+$, if $\ell(s) = s'' | {\downarrow}\langle s' \rangle$ and

  $$\ell(s') = a(z).s_a + b(z).s_b + \mathsf{nil}.s_{\mathsf{nil}} + \cdots,$$

  we have transitions

  $$s \xrightarrow{a} s_a, \qquad s \xrightarrow{b} s_b.$$

  ($\ell(s'')$ is always a variable by standardness.) We also let $s \in Q_F$ iff $\ell(s_{\mathsf{nil}}) = \maltese$.
- Otherwise, $\ell(s)$ is one of Ω, $s'' | \overline{c}\langle s'_1, \ldots, s'_k \rangle$ with $\overline{c} \neq {\downarrow}$ and $\ell(s'') = y$, and $\maltese$. In the first two cases, $\mathsf{design}(G, s)[w^\star/y] \Uparrow$ for any word w. Hence we let

  $$s \xrightarrow{a} s, \quad s \xrightarrow{b} s, \quad s \notin Q_F,$$

  meaning that the automaton accepts no inputs once s is visited. In the last case $\ell(s) = \maltese$, $\mathsf{design}(G, s)[w^\star/y] \Downarrow \maltese$ for any word w. Hence we let

  $$s \xrightarrow{a} s, \quad s \xrightarrow{b} s, \quad s \in Q_F,$$

  meaning that the automaton accepts any input once s is visited.

It is easy to see that for any $w \in \Sigma^*$, M accepts w iff $P[w^\star/x] \Downarrow \maltese$. □

Remark 2.24 The above theorem is specific to DFAs on *words*. There does not seem to be a canonical way to encode automata on *trees* as standard c-designs. Moreover, the argument for the second claim works just because we have restricted the inputs to words. Acceptance of trees would be naturally explained if we adopt a more parallel notion of designs, like L-nets of [6].

As we have pointed out, the reduction-based normalization procedure given in subsection 2.3 is not quite satisfactory, because it involves substitution and renaming, and so does not directly work on generators. Here we present another normalization procedure in the style of Krivine's abstract machine [15]. It works on generators, hence provides an effective means of normalization for finitely generated c-designs. Similar procedures are given by Faggian [4] and Curien [2]. However, unlike the token machine in the latter, our machine employs nesting of closures and environments to properly deal with bound variables (rather than absolute addresses).

Throughout this section, we fix a pointed generator $G = (S^+, S^-, \ell, s_I)$.

Definition 2.25 The set of closures and that of environments are defined by simultaneous induction. A *closure* c is a pair $s\rho$ of $s \in S = S^+ \cup S^-$ and an environment ρ. An *environment* ρ is a finite set $\{(x_1, c_1), \cdots, (x_n, c_n)\}$ such that $x_1, \ldots, x_n$ are distinct variables and $c_1, \ldots, c_n$ are closures. We denote it by $[x_1 \mapsto c_1, \ldots, x_n \mapsto c_n]$. If $\rho = [x_1 \mapsto c_1, \ldots, x_n \mapsto c_n]$, $\rho(x_i)$ stands for c_i. $\rho[x_i \mapsto c'_i]$ is the same as ρ except that $\rho[x_i \mapsto c'_i](x_i) = c'_i$. The empty environment is written as $[]$.

A *positive configuration* $(s\rho)$ simply consists of a closure $s\rho$ with $s \in S^+$, while a *negative configuration* is a pair $(s\rho, \phi)$ such that $s \in S^-$ and ϕ (which corresponds to the stack of Krivine's abstract machine) is a positive action followed by a finite list of closures: $\phi = \overline{a}\langle c_1, \ldots, c_n\rangle$ with $n = ar(a)$.

The procedure starts by the *initial configuration* $(s_I[])$, and follows the transition rules below. The transition relation between two states is denoted by $\longrightarrow$. For simplicity of description, we confuse a state s with its label $\ell(s)$.

$$
\begin{array}{rcl}
(\maltese\rho) & & \text{terminates;} \\
(\Omega\rho) & & \text{diverges;} \\
(s_0|\overline{a}\langle s_1, \ldots, s_n\rangle\ \rho) & \longrightarrow & (s_0\rho, \overline{a}\langle s_1\rho, \ldots, s_n\rho\rangle) \\
((\sum a(\vec{x}_a).s_a)\rho,\ \overline{a}\langle c_1, \ldots, c_n\rangle) & \longrightarrow & (s_a\rho[x_1 \mapsto c_1, \ldots, x_n \mapsto c_n]) \\
(x\rho, \phi) & \longrightarrow & (\rho(x), \phi), \quad \text{if } \rho(x) \text{ is defined.}
\end{array}
$$

In particular when $\ell(s) = s_0|\overline{a}\langle s_1, \ldots, s_n\rangle$ with $\ell(s_0) = x$, we have

$$(s\rho) \longrightarrow (x\rho,\ \overline{a}\langle s_1\rho, \ldots, s_n\rho\rangle) \longrightarrow (\rho(x),\ \overline{a}\langle s_1\rho, \ldots, s_n\rho\rangle),$$

which can be considered as a single step.

The above procedure works for closed c-designs. Applied to an open one, it may get stuck at $(x\rho, \phi)$ with $\rho(x)$ undefined (x is then construed as the 'head variable' of the normal form). Although it is possible to extend the procedure to open c-designs, we prefer to delegate it to the subsequent work.

One can verify that $(s_I[]) \longrightarrow^* (\maltese\rho)$ if and only if $\mathsf{design}(G, s_I) \Downarrow \maltese$. Moreover, the computation is effective. Hence when restricted to computation over word designs, we have the following:

Theorem 2.26 *Let Σ be an alphabet. For every finitely generated positive c-design P, there exists a Turing machine M such that*

(*) for any word $w \in \Sigma^*$, M accepts w if and only if $P[w^\star/x] \Downarrow \maltese$.

Proof There is a finite pointed generator (G, s_I) for P. One can define a Turing machine M which, given a word $w \in \Sigma^*$ as input, yields a finite generator (G', s'_I) for $P[w^\star/x]$ and then applies the Krivine style normalization procedure. M terminates if and only if the procedure does. $\square$

The converse will be taken up in subsection 3.1.

3 Analytical Theorems

The designs of ludics enjoy a number of fundamental properties, called *analytical theorems* in [9]. In this section, we reprove some of them in our new setting with special emphasis on their relevance to computational issues.

Associativity (subsection 3.1) is a weak form of the confluence property. It guarantees that composition of function designs works as expected. Monotonicity (subsection 3.2) states that normalization preserves natural orderings of c-designs.

Separation (subsection 3.3) is an analogue of Böhm's theorem in lambda calculus, meaning that two distinct standard c-designs can be separated via interaction with another c-design. We prove a stronger form of this property for data designs, which can be intuitively understood as saying that one can associate to each finite data design d a counter design (or a "machine") which accepts d and (essentially) nothing else. This is obvious for designs representing words, but we prove it for arbitrary finite data designs.

Finally, the pull-back property (subsection 3.4) informally states that linear c-designs are truly linear (in the sense of coherent semantics). It implies that merging of counter designs has a desired effect, and thus leads to a separation result for sets of finite data designs. The pull-back property also implies stability, just as linearity of a map in coherent semantics implies its stability.

3.1 Associativity

The first property to be stated is a limited form of confluence.

Theorem 3.1 (Associativity) *Let T be a c-design and $N_1, \ldots, N_n$ be negative c-designs. Then,*

$$[\![\; T[N_1/y_1, \ldots, N_n/y_n] \;]\!] = [\![\; [\![T]\!][\; [\![N_1]\!]/y_1, \ldots, [\![N_n]\!]/y_n] \;]\!].$$

It is intuitively clear that it holds, since our c-designs reasonably generalize Girard's original designs and lambda terms, both enjoying associativity. A formal proof is however lengthy, and is postponed to our subsequent work due to lack of space.

An immediate consequence of associativity is that function designs compose naturally.

Lemma 3.2 *Let $F[x]$ and $G[y]$ be function designs and d_0 a data design. If $[\![F[d_0]]\!] = d_1$ and $[\![G[d_1]]\!] = d_2$, then $[\![G[F[d_0]]]\!] = d_2$. The same holds for composition of multi-arity functions.*

We can now prove the converse of Theorem 2.26.

Theorem 3.3 *Let Σ be an alphabet. For every Turing machine M, there exists a finitely generated positive linear c-design $\mathsf{M}^\star$ without identities such that*

(*) for any $w \in \Sigma^*$, M accepts w if and only if $\mathsf{M}^\star[w^\star/x_0] \Downarrow \maltese$.

Proof (Sketch) Given a Turing machine M, it is routine to build a c-design $\mathsf{M}^\star$ which satisfies (*) by using constructors, discriminators, duplicators in subsection 2.4, together with $P[x] = x| \downarrow \langle \overline{\mathsf{zero}}.\maltese \rangle$, and applying composition and general recursion. Here $P[x]$ is used to turn a function design into a positive c-design which converges or diverges depending on the output: for any data design d, $P[d] \Downarrow \maltese$ if and only if $d = 0^\star$.

These are all linear and finitely generated. Identities can be removed by Proposition 2.15 (see also Remark 2.18). Moreover, composition of two function designs preserves finiteness of generators by Proposition 2.11, and yields an expected function design by Lemma 3.2. The same holds for general recursion (Proposition 2.19). Therefore, the resulting c-design $\mathsf{M}^\star$ is linear, identity-free and finitely generated. □

Designs admit two orderings. The first one, stable ordering $\sqsubseteq$, is an analogue of Berry's ordering in domain theory (see, eg., [10]). It captures the degree of superimposition: $T \sqsubseteq U$ means that U is 'more superimposed,' or 'more defined' than T.

Definition 3.4 The *stable ordering* $\sqsubseteq$ is the largest binary relation R over c-designs such that

(1) if $\maltese\ R\ T$ then $T = \maltese$;
(2) if $\Omega\ R\ T$, then T is positive;
(3) if $N_0|\overline{a}\langle N_1, \ldots, N_n\rangle\ R\ T$ then $T = M_0|\overline{a}\langle M_1, \ldots, M_n\rangle$ and $N_i\ R\ M_i$ for every $0 \leq i \leq n$;
(4) if $x\ R\ T$ then $T = x$;
(5) if $(\sum a(\vec{x}_a).P_a)\ R\ T$ then $T = \sum a(\vec{x}_a).Q_a$ and $P_a\ R\ Q_a$ for every $a \in A$.

The second one, observational ordering $\preceq$, is an analogue of the standard extensional ordering in domain theory. As we shall see, it corresponds to the likelihood of convergence: $T \preceq U$ means that U is more likely to converge than T when interacting with other designs.

Definition 3.5 The *observational ordering* $\preceq$ is the largest binary relation R over c-designs that satisfies (1), (2), (4), (5) of Definition 3.4 and

(3') if $N_0|\overline{a}\langle N_1, \ldots, N_n\rangle\ R\ T$ then $T = \maltese$ or $T = M_0|\overline{a}\langle M_1, \ldots, M_n\rangle$ and $N_i\ R\ M_i$ for every $0 \leq i \leq n$.

It is clear that both $\sqsubseteq$ and $\preceq$ are partial orderings, and $T \sqsubseteq U$ implies $T \preceq U$: *more defined, more likely to converge.* An impressive inequality is

$$\Omega \sqsubseteq P \preceq \maltese,$$

which holds for any positive c-design P. Since the only difference between $\sqsubseteq$ and $\preceq$ lies in the treatment of $\maltese$, two orderings coincide on the set of $\maltese$-free c-designs.

Any pair of distinct data designs is incomparable with respect to $\sqsubseteq$ (and $\preceq$, which coincides with $\sqsubseteq$ over the data designs). Hence $d \sqsubseteq e$ implies $d = e$ for any data designs d and e.

We now show that substitution and normalization are monotone with respect to these orderings. In particular, it confirms our intuition that the ordering $\preceq$ captures likelihood of convergence.

Theorem 3.6 (Monotonicity)

(1) If $T \preceq U$ and $M \preceq N$, then $T[M/x] \preceq U[N/x]$.
(2) If $T \preceq U$, then $[\![T]\!] \preceq [\![U]\!]$.

The same holds for the stable ordering $\sqsubseteq$.

Proof (1) Define a binary relation R by

- $T_0 \; R \; U_0 \iff T_0 = T_1[M_1/x]$ and $U_0 = U_1[N_1/x]$ for some T_1, U_1, M_1, N_1 such that $T_1 \preceq U_1$ and $M_1 \preceq N_1$.

One can easily verify that R satisfies (1), (2), (3'), (4), (5) of Definitions 3.4 and 3.5. Since $T[M/x] \; R \; U[N/x]$, we conclude $T[M/x] \preceq U[N/x]$.

(2) We first prove the following statement:

(*) If $P \preceq Q$ and $P \Downarrow P_0$, then $Q \Downarrow Q_0$ and $P_0 \preceq Q_0$.

The proof proceeds by induction on the length of the reduction sequence $P \longrightarrow^* P_0$.

When $P = P_0$, the claim is trivial. Otherwise, P is of the form $(\sum a(\vec{x}_a).P_a) \,|\overline{a}\langle\vec{M}\rangle$, which reduces to $P_a[\vec{M}/\vec{x}_a]$. Since $P \preceq Q$, Q is either $\maltese$ or of the form $(\sum a(\vec{x}_a).Q_a) \,|\overline{a}\langle\vec{N}\rangle$, where $P_a \preceq Q_a$ for every $a \in A$ and $\vec{M} \preceq \vec{N}$.

In the former case, we have $P_a[\vec{M}/\vec{x}_a] \preceq \maltese$. In the latter case, Q reduces to $Q_a[\vec{N}/\vec{x}_a]$ and we have $P_a[\vec{M}/\vec{x}_a] \preceq Q_a[\vec{N}/\vec{x}_a]$ by (1) above. In any case, the induction hypothesis applies, and we conclude (*).

Let us now define a binary relation R by

- $T_0 \; R \; U_0 \iff T_0 = [\![T_1]\!]$ and $U_0 = [\![U_1]\!]$ for some T_1, U_1 such that $T_1 \preceq U_1$.

Then R satisfies the properties (1), (2), (3'), (4), (5).

For instance, let us verify (3'). Assume $N_0|\overline{a}\langle N_1, \ldots, N_n\rangle \; R \; T$. Then by definition, there are P and Q such that $[\![P]\!] = N_0|\overline{a}\langle N_1, \ldots, N_n\rangle$, $[\![Q]\!] = T$ and $P \preceq Q$. This means that $N_0 = x$ and there are $M_1, \ldots, M_n$ such that $P \Downarrow x|\overline{a}\langle M_1, \ldots, M_n\rangle$ and $[\![M_i]\!] = N_i$ for $1 \leq i \leq n$. By (*) above and the definition of $\preceq$, either $Q \Downarrow \maltese$, or $Q \Downarrow x|\overline{a}\langle L_1, \ldots, L_n\rangle$ and $M_i \preceq L_i$ for $1 \leq i \leq n$. In the former case, we have $T = [\![Q]\!] = \maltese$. In the latter case, $T = x|\overline{a}\langle [\![L_1]\!], \ldots, [\![L_n]\!]\rangle$. Since $x \; R \; x$ and $N_i = [\![M_i]\!] \; R \; [\![L_i]\!]$, (3') holds.

Now if $T \preceq U$, $[\![T]\!] \; R \; [\![U]\!]$. Therefore we conclude $[\![T]\!] \preceq [\![U]\!]$. $\square$

We first define the orthogonality relation between c-designs (and anti-designs), and then discuss the separation property as well as a stronger form of it. In the sequel, we fix a variable x_0, which plays the role of the absolute address for atomic positive c-designs.

Definition 3.7 Let P and N be positive and negative c-designs respectively. P is said to be *closed* if it has no free variables. This implies that P is Ω, ✠ or a cut. P is *atomic* if $\mathsf{fv}(P) \subseteq \{x_0\}$, and N is *atomic* if $\mathsf{fv}(N) = \emptyset$. Two atomic c-designs P, N of opposite polarities are said to be *orthogonal* and written $P \perp N$ if $P[N/x_0] \Downarrow$ ✠.

It is possible to extend orthogonality to arbitrary c-designs. For that, we need the notion of anti-designs.

Definition 3.8 An *anti-design against positives* is a set $\{(x_1, N_1), \ldots, (x_n, N_n)\}$ where $x_1, \ldots, x_n$ are distinct variables and $N_1, \ldots, N_n$ are atomic negative c-designs. We denote it by $[N_1/x_1, \ldots, N_n/x_n]$. A positive c-design P and $[G] = [N_1/x_1, \ldots, N_n/x_n]$ are said to be *orthogonal* and written $P \perp [G]$ if the result of substitution $P[N_1/x_1, \ldots, N_n/x_n]$ is closed and converges to ✠.

An *anti-design against negatives* is a set $\{P, (x_1, N_1), \ldots, (x_n, N_n)\}$ where $\{(x_1, N_1), \ldots, (x_n, N_n)\}$ is as above and P is an atomic positive c-design. We denote it by $[P, N_1/x_1, \ldots, N_n/x_n]$. A negative c-design M and $[G] = [P, N_1/x_1, \ldots, N_n/x_n]$ are said to be *orthogonal* and written $M \perp [G]$ if $P\Big[M[N_1/x_1, \ldots, N_n/x_n]/x_0\Big]$ is closed and converges to ✠. In the sequel, we use notations $[G], [H], \ldots$ to denote arbitrary anti-designs of both polarities.

We say that an anti-design is total (resp. linear, cut-free, identity-free or standard) if its component c-designs are.

Theorem 3.6 (monotonicity) entails that if $T \preceq U$ and $T \perp [G]$ then $U \perp [G]$ for any anti-design $[G]$ against the polarity of T and U. The *separation property* is concerned with the converse. It holds for standard c-designs:

Theorem 3.9 (Separation) *If T and U are standard and $T \not\preceq U$, then there is a standard anti-design $[G]$ such that $T \perp [G]$ and $U \not\perp [G]$.*

See [9] or [4] for a proof. As a consequence, we have $T = U$ if and only if $T \perp [G] \iff U \perp [G]$ for any $[G]$. Hence the internal structure of a standard c-design can be completely determined by its external behaviour. Notice that this does not hold for c-designs with cuts and/or identities, and non-linear c-designs.

In the above statement of the separation property, the anti-design $[G]$ separating T and U depends on both T and U. On the other hand, it is also possible to consider a notion of separation for which the separating anti-design depends only on T.

Definition 3.10 A standard c-design T admits *strong separation* if

- there is an anti-design $[T^c]$ such that $T \perp [T^c]$ and $U \not\perp [T^c]$ for any standard $\maltese$-*free* c-design U such that $T \not\preceq U$.

Here we restrict ourselves to $\maltese$-free U for a practical reason; without this restriction, very few c-designs would admit strong separation.

Our aim here is not to get into a general study of strong separation, but to exhibit how it is useful in analysis of computation. Hence we focus on the data designs and show that all *finite* ones enjoy the strong separation property. For that, we first define the *counter design* $(d)^c_{x_0}$ for each finite data design d.

Definition 3.11 Given a finite data design d and a negative c-design N, we define a positive c-design $(d)^c_N$ by induction on the structure of d:

$$\begin{aligned}(d)^c_N &= N|{\downarrow}\langle b.\maltese\rangle && \text{if } d = {\uparrow}\overline{b},\\ &= N|{\downarrow}\langle a(x_1,\dots,x_n).(d_1)^c_{x_1}[(d_2)^c_{x_2}/\maltese]\cdots[(d_n)^c_{x_n}/\maltese]\rangle && \text{if } d = {\uparrow}\overline{a}\langle d_1,\dots,d_n\rangle\end{aligned}$$

with $n \geq 1$, where $P[Q/\maltese]$ is obtained by replacing all occurrences of $\maltese$ in P by Q.

For instance, if $d = {\uparrow}\overline{a}\langle{\uparrow}\overline{b},{\uparrow}\overline{c}\rangle$, then $(d)^c_{x_0} = x_0|{\downarrow}\langle a(x_1,x_2).x_1|{\downarrow}\langle b.x_2|{\downarrow}\langle c.\maltese\rangle\rangle\rangle$. Note that every $(d)^c_{x_0}$ constructed this way is standard and has exactly one occurrence of $\maltese$.

Theorem 3.12 (Strong separation for data designs) *Let d be a finite data design. For any negative standard c-design N which is $\maltese$-free, $(d)^c_{x_0} \perp N$ if and only if $d \preceq N$.*

In the above statement, $d \preceq N$ can be replaced with $d \sqsubseteq N$, since N is $\maltese$-free.

Proof As to the 'if' direction, one can easily observe that $(d)^c_{x_0} \perp d$. Hence by monotonicity, $(d)^c_{x_0} \perp N$.

The converse direction is proved by induction on d.

Suppose that $d = \uparrow\overline{b}$. If $(d)^c_{x_0}[N/x_0] = (d)^c_N$ converges, then N must be $\uparrow\overline{b}+K$, where K is of the form $\sum_{A\setminus\{\uparrow\}} a(\vec{y}_a).P_a$, so that we have

$$(d)^c_N = (\uparrow\overline{b} + K)|\downarrow\langle b.\maltese\rangle \longrightarrow (b.\maltese)|\overline{b} \longrightarrow \maltese.$$

Hence we have $d \preceq N$.

Now suppose that $d = \uparrow \overline{a}\langle d_1, \ldots, d_n\rangle$ with $n \geq 1$. If $(d)^c_{x_0}[N/x_0] = (d)^c_N$ converges, then N must be of the form $\uparrow\overline{a}\langle N_1, \ldots, N_n\rangle + K$ so that we have

$$\begin{aligned}
(d)^c_N \;&=\; N|\downarrow\Big\langle a(x_1,\ldots,x_n).(d_1)^c_{x_1}[(d_2)^c_{x_2}/\maltese]\cdots[(d_n)^c_{x_n}/\maltese]\Big\rangle\\
&\longrightarrow \Big(a(x_1,\ldots,x_n).(d_1)^c_{x_1}[(d_2)^c_{x_2}/\maltese]\cdots[(d_n)^c_{x_n}/\maltese]\Big)|\overline{a}\langle N_1,\ldots,N_n\rangle\\
&\longrightarrow (d_1)^c_{N_1}[(d_2)^c_{N_2}/\maltese]\cdots[(d_n)^c_{N_n}/\maltese],
\end{aligned}$$

and the last one converges. Notice that

$$(d_1)^c_{N_1} \succeq (d_1)^c_{N_1}[(d_2)^c_{N_2}/\maltese] \succeq \cdots \succeq (d_1)^c_{N_1}[(d_2)^c_{N_2}/\maltese]\cdots[(d_n)^c_{N_n}/\maltese].$$

Hence $(d_1)^c_{N_1}$ also converges by Theorem 3.6(2). Since N_1 is $\maltese$-free, it converges just because the normalization visits the occurrence of $\maltese$ in $(d_1)^c_{x_1}$. In conjunction with the convergence of

$$(d_1)^c_{N_1}[(d_2)^c_{N_2}/\maltese]\cdots[(d_n)^c_{N_n}/\maltese] = (d_1)^c_{N_1}\left[\Big((d_2)^c_{N_2}[(d_3)^c_{N_2}/\maltese]\cdots[(d_n)^c_{N_n}/\maltese]\Big)/\maltese\right]$$

we see that $(d_2)^c_{N_2}[(d_3)^c_{N_3}/\maltese]\cdots[(d_n)^c_{N_n}/\maltese]$ converges too. By repetition, we see that $(d_k)^c_{N_k}$ converges for every $1 \leq k \leq n$. By induction hypothesis, $d_k \preceq N_k$. Hence $d \preceq N$. $\square$

Remark 3.13 Our notion of strong separation is closely related to the notion of interactive observability studied by Faggian [4]. In fact, it is possible to construe Theorem 3.12 as a special case of the characterization of interactive observability of slices via counter-slices in [4].

3.4 Compatibility and stability

In the previous subsection, we have shown that each finite data design can be strongly separated. In applications, however, it is more important to separate each *set* $\mathbf{D}$ of finite data designs from others. For that, we need to find a counter design which works for all elements of $\mathbf{D}$. Here the key operation is to merge the counter designs $\{(d)^c_{x_0} | d \in \mathbf{D}\}$. We therefore introduce the union and intersection operations on c-designs.

Definition 3.14 The *union* $T \cup U$ of two c-designs T, U is defined as a partial operation:

- $\maltese \cup \maltese = \maltese$;
- $P \cup \Omega = \Omega \cup P = P$;
- $N_0|\overline{a}\langle N_1, \dots, N_n\rangle \cup M_0|\overline{a}\langle M_1, \dots, M_n\rangle = N_0 \cup M_0|\overline{a}\langle N_1 \cup M_1, \dots, N_n \cup M_n\rangle$ if $N_0 \cup M_0, \dots, N_n \cup M_n$ are defined;
- $x \cup x = x$;
- $\sum a(\vec{x}_a).P_a \cup \sum a(\vec{x}_a).Q_a = \sum a(\vec{x}_a).(P_a \cup Q_a)$ if $P_a \cup Q_a$ is defined for every $a \in A$;
- $T \cup U$ is not defined otherwise.

The *intersection* $T \cap U$ can be defined in the almost same way as union, except that

- $P \cap \Omega = \Omega \cap P = \Omega$.

T and U are *compatible* if there is a c-design V such that $T \sqsubseteq V$ and $U \sqsubseteq V$.

Formally, unions and intersections are defined by an extension of the corecursion principle (Theorem 2.12) to partial functions. Although only binary unions and intersection are defined above, they can be extended to arbitrary ones without any problem.

Lemma 3.15 *(1) If T and U are compatible, so are $[\![T]\!]$ and $[\![U]\!]$.*
(2) T and U are compatible iff $T \cup U$ is defined iff $T \cap U$ is defined.

Two distinct data designs are never compatible. Hence one cannot take the union. On the contrary, we have:

Lemma 3.16 *For any finite data designs d and e, $(d)^c_y$ and $(e)^c_y$ are compatible.*

This is intuitively clear, as the only positive actions in $(d)^c_y$ and $(e)^c_y$ are $\downarrow$ and $\maltese$. Hence there are very few chances of conflict. A formal proof is as follows.

Proof We show the following statement by induction on the structure of d:

- for any data design e and any compatible pair (P, Q) of positive c-designs, $(d)^c_y[P/\maltese]$ and $(e)^c_y[Q/\maltese]$ are compatible.

The lemma then follows by taking $P = Q = \maltese$.

If $d = \uparrow\overline{b}$, then $(d)^c_y[P/\maltese] = y|\downarrow\langle b.P\rangle$. If e is also $\uparrow\overline{b}$, then the claim is trivial. Otherwise, $(e)^c_y[Q/\maltese]$ is of the form $y|\downarrow\langle c(\vec{x}_c).R\rangle$ with $c \neq b$. Hence one can take the union $y|\downarrow\langle b.P + c(\vec{x}_c).R\rangle$.

If $d = \uparrow\overline{a}\langle d_1, \ldots, d_n\rangle$, then $(d)^c_y[P/\maltese]$ is of the form

$$y|\downarrow\langle a(x_1, \ldots, x_n).(d_1)^c_{x_1}[(d_2)^c_{x_2}/\maltese] \cdots [(d_n)^c_{x_n}/\maltese][P/\maltese]\rangle.$$

If e is of the form $\uparrow\overline{c}\langle\vec{e}\rangle$ with $c \neq a$, then the proof proceeds as in the previous case. So suppose that $e = \uparrow\overline{a}\langle e_1, \ldots, e_n\rangle$. Then $(e)^c_y[Q/\maltese]$ is of the form

$$y|\downarrow\langle a(x_1, \ldots, x_n).(e_1)^c_{x_1}[(e_2)^c_{x_2}/\maltese] \cdots [(e_n)^c_{x_n}/\maltese][Q/\maltese]\rangle.$$

By induction hypothesis, $(d_n)^c_{x_n}[P/\maltese]$ and $(e_n)^c_{x_n}[Q/\maltese]$ are compatible. Hence so are $(d_{n-1})^c_{x_{n-1}}[(d_n)^c_{x_n}/\maltese][P/\maltese]$ and $(e_{n-1})^c_{x_{n-1}}[(e_n)^c_{x_n}/\maltese][Q/\maltese]$ (note that the former can also be written as $(d_{n-1})^c_{x_{n-1}}\left[\left((d_n)^c_{x_n}[P/\maltese]\right)/\maltese\right]$ and similarly for the latter). By repetition, we see that $(d)^c_y[P/\maltese]$ and $(e)^c_y[Q/\maltese]$ are compatible. $\square$

Lemmas 3.16 and 3.15(2) allows us to define a counter design for a set of data designs.

Definition 3.17 Given a nonempty set $\mathbf{D}$ of finite data designs, we define

$$c(\mathbf{D}) = \bigcup\{(d)^c_{x_0} : d \in \mathbf{D}\} \cup \{x_0|\downarrow\langle 0\rangle\},$$

where $x_0|\downarrow\langle 0\rangle$ is added just in case of $\mathbf{D}$ being empty.

To establish that this $c(\mathbf{D})$ works as a separator for the set $\mathbf{D}$, we have to show that $c(\mathbf{D})\perp N$ for a standard $\maltese$-free N implies $(d)^c_{x_0}\perp N$ for some $d \in \mathbf{D}$; then we would be able to conclude $d \preceq N$ by Theorem 3.12. Although it is possible to prove it directly, we prefer to derive it from a more general principle. That is nothing but the pull-back property of [9].

Definition 3.18 A (finite) *slice* is a finite c-design in which all negative subdesigns are either 0 or of the form $a(\vec{x}_a).P_a$ (i.e., at most unary branching). U is a *slice of* T if U is a slice and $U \sqsubseteq T$.

All data designs are slices by definition.

The notion of slice is useful to analyze the structure of *linear* c-designs in two ways. First, linearity of a c-design T implies that every variable occurs at most once in every slice of T. Second, normalization of a linear c-design is performed slice-wise. The pull-back property formalizes this second aspect.

Theorem 3.19 (Pull-back) *Let T be a linear c-design. For any slice U' of*

$U = [\![T]\!]$, there exists a unique minimal slice T' of T such that $[\![T']\!] = U'$:

$$\begin{tikzcd}
T \arrow[r] & U \\
T' \arrow[u, dotted, "slice_of"] \arrow[r, dotted] & U' \arrow[u, "slice_of"']
\end{tikzcd}$$

Proof First of all, consider the following reduction:

$$P = (\sum a(\vec{x}_a).P_a)\,|\overline{a}\langle\vec{N}\rangle \longrightarrow P_a[\vec{N}/\vec{x}_a] = Q,$$

where $\vec{x}_a = x_1, \ldots, x_n$ and $\vec{N} = N_1, \ldots, N_n$. Let Q' be a slice of Q. It is of the form $P'_a[\vec{N}'/\vec{x}_a]$, where P'_a, $\vec{N}'$ are respectively slices of $P_a, \vec{N}$. We assume that $\vec{N}' = N'_1, \ldots, N'_n$ are chosen minimal; namely $N'_i = 0$ if $x_i \notin \mathsf{fv}(\mathsf{P}'_\mathsf{a})$. From this, we obtain a slice $P' = a(\vec{x}_a).P'_a\ |\overline{a}\langle\vec{N}'\rangle$ of P:

$$\begin{tikzcd}
(\sum a(\vec{x}_a).P_a)\,|\overline{a}\langle\vec{N}\rangle \arrow[r] & P_a[\vec{N}/\vec{x}_a] \\
a(\vec{x}_a).P'_a\ |\overline{a}\langle\vec{N}'\rangle \arrow[u, dotted, "slice_of"] \arrow[r, dotted] & P'_a[\vec{N}'/\vec{x}_a] \arrow[u, "slice_of"']
\end{tikzcd}$$

Obviously P' is the minimal slice such that $P' \longrightarrow Q'$. This can be extended to the case when $P \longrightarrow^* Q \neq \Omega$ by induction on the length of the reduction sequence.

Now, the theorem is proved by induction on the structure of the slice U' (which is finite) of the normal form $U = [\![T]\!]$.

If $U' = \Omega$, then one can take $T' = \Omega$. If $U' = \maltese$, then the above argument yields the desired slice T' such that $T' \longrightarrow \maltese$. If $U' = x$, then $U = T = x$. Hence one can take $T' = x$.

If U' is of the form $x|\overline{a}\langle N'_1, \ldots, N'_n\rangle$, then U is of the form $x|\overline{a}\langle N_1, \ldots, N_n\rangle$ so that N'_i is a slice of N_i for $1 \leq i \leq n$. Since $U = [\![T]\!]$, we have $T \Downarrow x|\overline{a}\langle M_1, \ldots, M_n\rangle$ and $[\![M_i]\!] = N_i$ for $1 \leq i \leq n$. By induction hypothesis, there is a unique minimal slice M'_i of M_i such that $[\![M'_i]\!] = N'_i$. Since $x|\overline{a}\langle M'_1, \ldots, M'_n\rangle$ is a slice of $x|\overline{a}\langle M_1, \ldots, M_n\rangle$, the desired slice T' of T is obtained by pulling back $x|\overline{a}\langle M'_1, \ldots, M'_n\rangle$ along the reduction sequence $T \longrightarrow^* x|\overline{a}\langle M_1, \ldots, M_n\rangle$ as above.

If U' is of the form $a(\vec{x}_a).Q'_a$, then U is of the form $\sum a(\vec{x}_a).Q_a$. Since $U = [\![T]\!]$, T must be of the form $\sum a(\vec{x}_a).P_a$ so that $[\![P_a]\!] = Q_a$. Since Q'_a is a slice of Q_a, the induction hypothesis yields a unique minimal slice P'_a of P_a. Then one can take $a(\vec{x}_a).P'_a$ as the desired slice of T. $\square$

We are now ready to prove a separation result for sets of finite data designs.

Theorem 3.20 (Strong separation for sets of finite data designs) *Let* $\mathbf{D}$ *be a set of finite data designs. For any negative standard c-design* N *which is* ✠*-free,* $c(\mathbf{D})\perp N$ *if and only if* $d \preceq N$ *for some* $d \in \mathbf{D}$.

Proof If $d \preceq N$ for some $d \in \mathbf{D}$, then $(d)^c_{x_0}\perp N$ by Theorem 3.12. Since $(d)^c_{x_0} \sqsubseteq c(\mathbf{D})$, we have $c(\mathbf{D})\perp N$ by monotonicity.

Conversely, suppose that $c(\mathbf{D})\perp N$, i.e., $[\![c(\mathbf{D})[N/x_0]]\!] =$ ✠. By the pull-back theorem, there are a slice P' of $c(\mathbf{D})$ and a slice N' of N such that $[\![P'[N'/x_0]]\!] =$ ✠, i.e., $P'\perp N'$.

We claim that P' is a finite chain (or a *chronicle* [9]). Indeed, P' does not branch at a positive subdesign because the only proper positive action in P' is $\downarrow$ that is unary. It does not branch at a negative subdesign either, because P' is a slice. Furthermore, P' is finite by the definition of slice.

As a consequence, P' is contained in one counter design $(d)^c_{x_0}$ for some $d \in \mathbf{D}$. By monotonicity, we have $(d)^c_{x_0}\perp N$. Hence by Theorem 3.12, we conclude $d \preceq N$. □

We end this subsection by proving another important consequence of the pull-back theorem: stability. To properly state and prove it, we need the following lemma.

Lemma 3.21 *If every slice of* T *is also a slice of* U*, then* $T \sqsubseteq U$.

Proof Define a binary relation R by $T\ R\ U \iff U$ contains all slices of T. One can then verify that R satisfies (1) – (5) of Definition 3.4. □

Corollary 3.22 (Stability) *Let* $\{T_i\}_{i\in\Lambda}$ *be a family of linear c-designs. If* $\{T_i\}_{i\in\Lambda}$ *are pairwise compatible, then* $[\![\bigcap_{i\in\Lambda} T_i]\!] = \bigcap_{i\in\Lambda}[\![T_i]\!]$.

Proof The inclusion $\sqsubseteq$ follows by monotonicity. To show the converse, let U' be a common slice of $[\![T_i]\!]$ for all $i \in \Lambda$. By the pull-back theorem, each T_i contains a minimal slice T_i' such that $[\![T_i']\!] = U'$.

We claim that $T_i' = T_j'$ for every $i, j \in \Lambda$. For that, notice that $\bigcup_{i\in\Lambda} T_i$ is a linear c-design, $[\![T_i]\!] \sqsubseteq [\![\bigcup_{i\in\Lambda} T_i]\!]$ by monotonicity, and hence U' is also a slice of $[\![\bigcup_{i\in\Lambda} T_i]\!]$. By the pull-back theorem again, $\bigcup_{i\in\Lambda} T_i$ contains a minimal slice T_0'. By minimality of T_0' and T_i', we have $T_0' = T_i'$ for all $i \in \Lambda$ as required.

Since $\bigcap_{i\in\Lambda} T_i$ contains the slice T_0', $[\![\bigcap_{i\in\Lambda} T_i]\!]$ contains U' by monotonicity. Therefore by Lemma 3.21, $\bigcap_{i\in\Lambda}[\![T_i]\!] \sqsubseteq [\![\bigcap_{i\in\Lambda} T_i]\!]$. □

4 Behaviours and internal completeness

We have studied the designs, which correspond to proofs in logic, terms in lambda calculus, strategies in game semantics, processes in concurrency, and data and machines in automata and computability theories. We now step up to a higher level construct: the *behaviours*. Behaviours correspond to interpretations of formulas, computability predicates in strong normalization proofs, semantic types (see, eg., [17]), and truth values in Krivine realizability [14].

After introduction of behaviours (subsection 4.1), we discuss how to construe a behaviour as a language. Since behaviours usually contain a lot of irrelevant elements, we need to purify them by *incarnation* (subsection 4.2).

We then introduce logical connectives as behaviour constructors (subsection 4.3). On the one hand, they allow us to build a logical system, such as *polarized linear logic* [16], upon ludics, although it is left to our subsequent work. On the other hand, they can be seen as a generalization of language operators (such as union, prefixing). *Internal completeness* of logical connectives is essential for both views (subsection 4.4). Finally we sketch how to construct languages by logical connectives and other operators (subsection 4.5).

4.1 Behaviours

In the rest of this paper, we restrict ourselves to a special class of c-designs.

Definition 4.1 An *l-design* T is a total, linear, identity-free c-design such that $\mathsf{fv}(T)$ is finite. An *anti-l-design* is an anti-design that consists of l-designs.

Thus the standard c-designs are exactly the cut-free l-designs. Non-linear c-designs and c-designs with identities will be studied in the subsequent work.

The orthogonality relation $\perp$ is defined in Definition 3.7. It naturally induces a construction of sets of l-designs as in phase semantics [8].

Definition 4.2 Given an l-design T and anti-l-design $[G]$, we define

$$\begin{aligned} T^{\perp} &= \{[G] : T \perp [G], [G] \text{ is an anti-l-design}\}, \\ [G]^{\perp} &= \{T : T \perp [G], T \text{ is an l-design}\}. \end{aligned}$$

These definitions extend to $\mathbf{T}^{\perp}$ and $\mathbf{G}^{\perp}$, where $\mathbf{T}$ and $\mathbf{G}$ are respectively a set of l-designs and a set of anti-l-designs of the same polarity.

The basic properties of orthogonality are as follows:

Lemma 4.3 *For any sets* $\mathbf{X}, \mathbf{Y}$ *of l-designs of the same polarity (or of anti-l-designs of the same polarity), the following hold:*

(1) $\mathbf{X} \subseteq \mathbf{Y}$ *implies* $\mathbf{Y}^{\perp} \subseteq \mathbf{X}^{\perp}$.
(2) $\mathbf{X} \subseteq \mathbf{X}^{\perp\perp}$.
(3) $\mathbf{X}^{\perp} = \mathbf{X}^{\perp\perp\perp}$.
(4) $\mathbf{X} \subseteq \mathbf{Y}^{\perp\perp}$ *implies* $\mathbf{X}^{\perp\perp} \subseteq \mathbf{Y}^{\perp\perp}$.
(5) $(\mathbf{X} \cup \mathbf{Y})^{\perp} = \mathbf{X}^{\perp} \cap \mathbf{Y}^{\perp}$.

Definition 4.4 A *behaviour* $\mathbf{T}$ is a set of l-designs of the same polarity that is equal to its biorthogonal: $\mathbf{T} = \mathbf{T}^{\perp\perp}$. $\mathbf{T}$ is positive or negative depending on the polarity of l-designs in it. $\mathbf{T}$ is *atomic* if all l-designs in it are.

By Lemma 4.3(3), any set of the form $\mathbf{G}^{\perp}$ is a behaviour (where $\mathbf{G}$ is a set of anti-l-designs). By (5), an intersection of behaviours is also a behaviour.

In general, the orthogonal $\mathbf{T}^{\perp}$ of a set $\mathbf{T}$ of l-designs consists of anti-l-designs. But when $\mathbf{T}$ is atomic, $\mathbf{T}^{\perp}$ can also be considered as a behaviour:

$$\mathbf{T}^{\perp} = \{U : \forall T \in \mathbf{T}.\ T \perp U,\ U \text{ is an l-design}\}.$$

There are the least and greatest atomic positive (resp. negative) behaviours $\mathbf{0}^{+}, \top^{+}$ (resp. $\mathbf{0}^{-}, \top^{-}$):

$$\begin{aligned}
\mathbf{0}^{+} &= \{\maltese\} &&= \{\text{atomic negative l-designs}\}^{\perp};\\
\top^{+} &= \{\text{atomic positive l-designs}\} &&= \emptyset^{\perp};\\
\mathbf{0}^{-} &= \{\maltese^{-}\} &&= \{\text{atomic positive l-designs}\}^{\perp};\\
\top^{-} &= \{\text{atomic negative l-designs}\} &&= \emptyset^{\perp};
\end{aligned}$$

where $\maltese^{-} = \sum a(\vec{x}_a).\maltese$.

Proposition 4.5 *Every behaviour* $\mathbf{T}$ *satisfies the following closure properties:*

- *Closure under the observational ordering:* $T \in \mathbf{T}$ *and* $T \preceq U$ *implies* $U \in \mathbf{T}$.
- *Closure under* β*-equivalence:* $T \in \mathbf{T}$ *iff* $[\![T]\!] \in \mathbf{T}$.
- *Closure under intersection: for any set* $\{T_i\}_{i\in\Lambda}$ *of compatible l-designs in* $\mathbf{T}$, $\bigcap_{i\in\Lambda} T_i \in \mathbf{T}$.

These properties are respectively due to monotonicity (Theorem 3.6), associativity (Theorem 3.1) and stability (Corollary 3.22).

Just as anti-designs are built from atomic c-designs, *anti-behaviours* are built from atomic behaviours.

Definition 4.6 Given an atomic positive behaviour $\mathbf{P}$, atomic negative behaviours $\mathbf{N}_1, \ldots, \mathbf{N}_n$ and distinct variables $x_1, \ldots, x_n$, we define

$$[\vec{\mathbf{N}}/\vec{x}] = \{[N_1/x_1, \ldots, N_n/x_n] : N_i \in \mathbf{N}_i \text{ for } 1 \le i \le n\};$$
$$[\mathbf{P}, \vec{\mathbf{N}}/\vec{x}] = \{[P, N_1/x_1, \ldots, N_n/x_n] : P \in \mathbf{P}, N_i \in \mathbf{N}_i \text{ for } 1 \le i \le n\},$$

where $\vec{\mathbf{N}}/\vec{x}$ stands for $\mathbf{N}_1/x_1, \ldots, \mathbf{N}_n/x_n$.

Observe that $P \in [\mathbf{N}/x]^\perp$ if and only if $P[x_0/x] \in \mathbf{N}^\perp$.

4.2 Incarnation

As Theorem 2.23 indicates, acceptance of a word w by a DFA M can be captured by orthogonality between $w^\star$ and $\mathsf{M}^\star$:

$$\mathsf{M} \text{ accepts } w \iff \mathsf{M}^\star \perp w^\star.$$

Hence one might expect that $\{\mathsf{M}^\star\}^\perp$ gives rise to (a representation of) the language accepted by M. However, this is not exactly the case, since the behaviour $\{\mathsf{M}^\star\}^\perp$ contains a lot of irrelevant elements. For instance, if $a^\star = \uparrow\overline{a}\langle\uparrow \mathsf{nil}\rangle \in \{\mathsf{M}^\star\}^\perp$, the following also belong to $\{\mathsf{M}^\star\}^\perp$ by Proposition 4.5:

- Any N such that $[\![N]\!] = a^\star$
- Any N of the form $\uparrow\overline{a}\langle\uparrow \mathsf{nil} + K_1\rangle + K_2$, where K_1 and K_2 are of the form $\sum_{A\setminus\{\uparrow\}} b(\vec{x}_b).P_b$.
- $\uparrow(x).\maltese$ and $\uparrow\overline{a}\langle\uparrow(x).\maltese\rangle$.

Hence to obtain a representation of a language, one has to remove these redundant l-designs from $\{\mathsf{M}^\star\}^\perp$. In [9], an operation called incarnation is introduced to remove the second type of redundancy. Roughly speaking, given an l-design U in a behaviour $\mathbf{T}$, the incarnation of U in $\mathbf{T}$ is the least portion of U that is required for interacting with the anti-l-designs in $\mathbf{T}^\perp$. Slightly deviating from [9], we also incorporate the effect of normalization into the definition to get rid of the first type of redundancy as well. The third one is removed by restricting l-designs to $\maltese$-free ones.

Definition 4.7 Let $\mathbf{T}$ be a behaviour and U an l-design in it. The *incarnation* of U in $\mathbf{T}$ is defined by

$$|U|_{\mathbf{T}} = \bigcap\{U' : U' \sqsubseteq [\![U]\!], U' \in \mathbf{T}\}.$$

An l-design U is *material* in $\mathbf{T}$ if $U = |U|_{\mathbf{T}}$. U is *pure* in $\mathbf{T}$ if it is material in $\mathbf{T}$ and furthermore $\maltese$-free. The set of all material (resp. pure) l-designs in $\mathbf{T}$ is denoted by $|\mathbf{T}|$ (resp. $||\mathbf{T}||$).

The incarnation $|U|_{\mathbf{T}}$ belongs to $\mathbf{T}$ due to stability (Proposition 4.5).

The set $|\mathbf{T}|$ in fact contains all necessary l-designs to interact with its opponents: $|\mathbf{T}|^{\perp} = \mathbf{T}^{\perp}$. In fact, $\mathbf{T}^{\perp} \subseteq |\mathbf{T}|^{\perp}$ by Lemma 4.3(1). To show the converse, let $[G] \in |\mathbf{T}|^{\perp}$ and $U \in \mathbf{T}$. Then $|U|_{\mathbf{T}} \in |\mathbf{T}|$ and hence $|U|_{\mathbf{T}} \perp [G]$. By monotonicity, $U \perp [G]$. Therefore $[G] \in \mathbf{T}^{\perp}$.

With the notions of incarnation and purity, we can give a purely ludics-theoretic definition of acceptance which applies to non-data designs as well.

Definition 4.8 Let T be an atomic l-design. A set $\mathbf{U}$ of l-designs is *accepted by* T if $||T^{\perp}|| = \mathbf{U}$.

Although the definition is general, we are mainly interested in the particular case of sets of data designs. The following lemma gives a sufficient condition for that.

Lemma 4.9 *Suppose that a positive l-design P satisfies the following property:*

- *For any $\maltese$-free negative cut-free l-design N such that $P \perp N$, there is a data design $d \sqsubseteq N$ such that $P \perp d$.*

Then $||P^{\perp}|| = \{d : P \perp d,\ d \textit{ is a data design}\}$.

Proof Suppose that $N \in ||P^{\perp}||$. Then N is cut-free, $\maltese$-free, and $P \perp N$. Hence by the condition there is a data design $d \sqsubseteq N$ such that $P \perp d$. Since N is material in $P^{\perp}$, we have $d = N$.

Conversely, assume that a data design d satisfies $P \perp d$. Recall that d is cut- and $\maltese$-free. If d is not material in $P^{\perp}$, there exists $N \not\sqsupseteq d$ such that $P \perp N$. Since N is also cut- and $\maltese$-free, the condition gives another data design $d' \sqsubseteq N \not\sqsupseteq d$, that is impossible (see subsection 3.2). Hence d is material in $P^{\perp}$, and in fact pure. $\square$

We are now ready to prove the main theorem of this paper, which illustrates the computational powers of arbitrary l-designs, finitely generated l-designs, and finitely generated cut-free l-designs.

Theorem 4.10

(1) Any set $\mathbf{D}$ *of finite data designs is accepted by an l-design.*

For any language $\mathbf{L} \subseteq \Sigma^*$,

(2) $\mathbf{L}^\star$ *is accepted by a finitely generated l-design if and only if* $\mathbf{L}$ *is recursively enumerable.*
(3) $\mathbf{L}^\star$ *is accepted by a finitely generated cut-free l-design if and only if* $\mathbf{L}$ *is regular.*

Proof (1) By Theorem 3.20, the positive l-design $c(\mathbf{D})$ satisfies the condition of Lemma 4.9. Hence $\mathbf{D} = ||c(\mathbf{D})^\perp||$.

(2) As to the 'if' direction, observe that the positive l-design $\mathsf{M}^\star$ in the proof of Theorem 3.3, when applied to a negative standard design N, only uses the data part $d \sqsubseteq N$ of N, since $\mathsf{M}^\star$ is obtained by composition and general recursion from constructors, discriminators, and duplicators and $P[x]$. Hence it satisfies the condition of Lemma 4.9. Moreover, $\mathsf{M}^\star$ can be built in such a way that it never accepts non-word data designs. Hence $\mathbf{L}^\star = ||\mathsf{M}^{\star\perp}||$. The converse direction immediately follows from Theorem 2.26.

(3) As to the 'if' direction, the positive cut-free l-design P in the proof of Theorem 2.23 satisfies the condition of Lemma 4.9 by definition. Hence $\mathbf{L}^\star = ||P^\perp||$. The converse direction immediately follows from the second statement of Theorem 2.23. $\square$

4.3 Logical connectives

In language and automata theory, there are basically two ways for defining a language.

- *By interaction*: give a machine or automaton and consider the set of words accepted by it.
- *By construction*: construct a language by various operators such as union, prefixing and Kleene's star.

Since behaviours generalize languages, it is natural to extend the above two approaches to definition of behaviours. The first approach, definition by interaction, has already been exploited: given an l-design T, take its orthogonal $T^\perp$ and then restrict it to the pure elements $||T^\perp||$. Although we have only discussed definition of behaviours that consist of data designs, this approach can be generalized to definition of arbitrary behaviours.

Now let us discuss the second approach, definition by construction. For that, the first thing to be observed is that some operations on languages do not generalize to behaviours. For instance, consider the union operation. Let $\mathbf{T}$ and $\mathbf{U}$ be two behaviours of the same polarity. Then it is not always the case that $\mathbf{T} \cup \mathbf{U}$ forms a behaviour. One can of course obtain the least behaviour that contains $\mathbf{T} \cup \mathbf{U}$ by taking biorthogonal: $(\mathbf{T} \cup \mathbf{U})^{\perp\perp}$. But then there is no guarantee that $||(\mathbf{T} \cup \mathbf{U})^{\perp\perp}|| = ||\mathbf{T}|| \cup ||\mathbf{U}||$; taking biorthogonal may add new pure elements. Hence a natural question is this: for which operation $\star$ on behaviours, do we have the property $||(\mathbf{T} \star \mathbf{U})^{\perp\perp}|| = ||\mathbf{T}|| \star ||\mathbf{U}||$?

In this subsection, we propose a definition of positive and negative logical connectives on behaviours, as analogues of language operators. They encompass connectives of polarized linear logic without exponentials [16]. In the next subsection, we show that all logical connectives enjoy *internal completeness*. In particular, all positive ones satisfy the above property, and thus can be used as language operators.

Given an m-ary name a and negative behaviours $\mathbf{M}_1, \dots, \mathbf{M}_m$, we define

$$\overline{a}\langle \mathbf{M}_1, \dots, \mathbf{M}_m \rangle = \{x_0 | \overline{a}\langle M_1, \dots, M_m\rangle : M_k \in \mathbf{M}_k \text{ for } 1 \leq k \leq m\}.$$

Given a set $\alpha = \{a(\vec{x}_a)\}_{a \in K}$ of negative actions (with $K \subseteq A$) and a positive behaviour $\mathbf{P}_a$ for each $a \in K$, we define

$$\textstyle\sum_\alpha a(\vec{x}_a).\mathbf{P}_a = \{\sum_K a(\vec{x}_a).P_a : P_a \in \mathbf{P}_a\}.$$

Definition 4.11 We presuppose a fixed ordering of variables other than x_0: $x_1, x_2, x_3, \dots$. An *n-ary logical connective* α is a finite set $\{a(\vec{x}_a)\}_{a \in K}$ of negative actions indexed by $K \subseteq A$ such that $\{\vec{x}_a\} \subseteq \{x_1, \dots, x_n\}$ for every $a \in K$. Given atomic negative behaviours $\mathbf{N}_1, \dots, \mathbf{N}_n$, and atomic positive behaviours $\mathbf{P}_1, \dots, \mathbf{P}_n$, we define

$$\begin{aligned}
\overline{\alpha}\langle \mathbf{N}_1, \dots, \mathbf{N}_n \rangle &= (\bigcup_{a(\vec{x}_a) \in \alpha} \overline{a}\langle \mathbf{N}_{i_1}, \dots, \mathbf{N}_{i_m}\rangle)^{\perp\perp}, \\
\alpha(\mathbf{P}_1, \dots, \mathbf{P}_n) &= (\overline{\alpha}\langle \mathbf{P}_1^{\perp}, \dots, \mathbf{P}_n^{\perp}\rangle)^{\perp},
\end{aligned}$$

where in the definition of $\overline{\alpha}\langle \mathbf{N}_1, \dots, \mathbf{N}_n\rangle$, indices $i_1, \dots, i_m$ vary for each $a(\vec{x}_a) \in \alpha$, and are determined by $\vec{x}_a = x_{i_1}, \dots, x_{i_m}$.

Two typical logical connectives are $\& = \{\pi_1(x_1), \pi_2(x_2)\}$ and $\parr = \{\wp(x_1, x_2)\}$. Let us write $\oplus = \overline{\&}$, $\iota_i = \overline{\pi}_i$, $\otimes = \overline{\parr}$, and $\bullet = \overline{\wp}$. We then have

$$\begin{aligned}
\oplus\langle \mathbf{N}, \mathbf{M}\rangle &= (\iota_1\langle \mathbf{N}\rangle \cup \iota_2\langle \mathbf{M}\rangle)^{\perp\perp}, & \&(\mathbf{P}, \mathbf{Q}) &= \iota_1\langle \mathbf{P}^{\perp}\rangle^{\perp} \cap \iota_2\langle \mathbf{Q}^{\perp}\rangle^{\perp}, \\
\otimes\langle \mathbf{N}, \mathbf{M}\rangle &= \bullet\langle \mathbf{N}, \mathbf{M}\rangle^{\perp\perp}, & \parr\langle \mathbf{P}, \mathbf{Q}\rangle &= \bullet\langle \mathbf{P}^{\perp}, \mathbf{Q}^{\perp}\rangle^{\perp}.
\end{aligned}$$

In [9], some remarkable completeness properties are proved. They are called *internal completeness*, because they can be stated and proved without recourse to any external entities such as syntax. We now prove our version of internal completeness.

A crucial role is played by the *counter designs*, which interactively determine the first action of their opponents (the same idea has already appeared in subsection 3.3). For every logical connective α, we define $\overline{\alpha}^c\langle \mathbf{N}_1, \ldots, \mathbf{N}_n\rangle = \bigcup_{a(\vec{x}_a)\in\alpha} \overline{a}\langle \mathbf{N}_{i_1}, \ldots, \mathbf{N}_{i_m}\rangle$. We also define $\alpha^c(\mathbf{P}_1, \ldots, \mathbf{P}_n)$ to be the set of l-designs of the form

$$a(\vec{x}_a).Q[x_{i_k}/x_0] + b_1(\vec{y}_{b_1}).\maltese + \cdots + b_l(\vec{y}_{b_l}).\maltese$$

where $a(\vec{x}_a) \in \alpha$, $\vec{x}_a = x_{i_1}, \ldots, x_{i_m}$, $1 \leq k \leq m$, $Q \in \mathbf{P}_{i_k}$ and $\alpha \backslash \{a(\vec{x}_a)\} = \{b_1(\vec{y}_{b_1}), \cdots, b_l(\vec{y}_{b_l})\}$. We abbreviate it by $a(\vec{x}_a).Q_{i_k}[x_{i_k}/x_0] + \maltese_\alpha$.

We also consider a weaker form of incarnation (*head incarnation*).

Definition 4.12 Given a positive behaviour $\mathbf{P}$, we define $|\mathbf{P}|_h$ to be the subset of $\mathbf{P}$ that consists of 'head normal' l-designs of the form $x_0|\overline{a}\langle \vec{M}\rangle$. Similarly, given a negative behaviour $\mathbf{N}$ and a logical connective α, $|\mathbf{N}|_\alpha$ is the subset of $\mathbf{N}$ that consists of 'head incarnated' l-designs of the form $\sum_\alpha a(\vec{x}_a).P_a$.

These operations are indeed incarnations at the head position, as will be witnessed by Corollary 4.15.

Lemma 4.13

(1) $|\alpha^c(\mathbf{N}_1^\perp, \ldots, \mathbf{N}_n^\perp)^\perp|_h \subseteq \bigcup_{a(\vec{x}_a)\in\alpha} \overline{a}\langle \mathbf{N}_{i_1}, \ldots, \mathbf{N}_{i_m}\rangle$.
(2) $\overline{a}\langle \mathbf{N}_1, \ldots, \mathbf{N}_n\rangle \subseteq \alpha^c(\mathbf{N}_1^\perp, \ldots, \mathbf{N}_n^\perp)^\perp$.
(3) $|\overline{\alpha}^c\langle \mathbf{P}_1^\perp, \ldots, \mathbf{P}_n^\perp\rangle^\perp|_\alpha \subseteq \sum_\alpha a(\vec{x}_a).[\mathbf{P}_{i_1}^\perp/x_{i_1}, \ldots, \mathbf{P}_{i_m}^\perp/x_{i_m}]^\perp$, *(where indices* $i_1, \ldots, i_m$ *depend on* $\vec{x}_a = x_{i_1}, \ldots, x_{i_m}$ *as before).*
(4) $\alpha(\mathbf{P}_1, \ldots, \mathbf{P}_n) = \overline{\alpha}^c\langle \mathbf{P}_1^\perp, \ldots, \mathbf{P}_n^\perp\rangle^\perp$.

Proof (1) Let $P = x_0|\overline{a}\langle M_1, \ldots, M_m\rangle$ be an l-design in $|\alpha^c(\mathbf{N}_1^\perp, \ldots, \mathbf{N}_n^\perp)^\perp|_h$. We see that $a(\vec{x}_a) \in \alpha$ for some $\vec{x}_a = x_1, \ldots, x_m$ because it is orthogonal to the l-designs in $\alpha^c(\mathbf{N}_1^\perp, \ldots, \mathbf{N}_n^\perp)$. Moreover, since P is orthogonal to $a(\vec{x}_a).Q[x_{i_k}/x_0] + \maltese_\alpha$ for any $1 \leq k \leq m$ and any $Q \in \mathbf{N}_{i_k}^\perp$, the reduction

$$(a(\vec{x}_a).Q[x_{i_k}/x_0] + \maltese_\alpha) \mid \overline{a}\langle M_1, \ldots, M_m\rangle \longrightarrow Q[x_{i_k}/x_0][\vec{M}/\vec{x}_a] = Q[M_k/x_0]$$

shows that $M_k \perp Q$, and so $M_k \in \mathbf{N}_{i_k}$. Hence we conclude $P \in \overline{a}\langle \mathbf{N}_{i_1}, \ldots, \mathbf{N}_{i_m}\rangle$.

(2) It is sufficient to show that $\bigcup_{a(\vec{x}_a)\in\alpha} \overline{a}\langle \mathbf{N}_{i_1}, \ldots, \mathbf{N}_{i_m}\rangle$ is a subset of the RHS by Lemma 4.3(3). So let P be of the form $x_0|\overline{a}\langle M_1, \ldots, M_m\rangle$ with $a(\vec{x}_a) \in \alpha$, $\vec{x}_a = x_{i_1}, \ldots, x_{i_m}$ and $M_k \in \mathbf{N}_{i_k}$ for every $1 \le k \le m$. Take N from $\alpha^c(\mathbf{N}_1^\perp, \ldots, \mathbf{N}_n^\perp)$ and check that N is orthogonal to P. The crucial case is $N = a(\vec{x}_a).Q[x_{i_k}/x_0] + \maltese_\alpha$ for some $1 \le k \le m$ and $Q \in \mathbf{N}_{i_k}^\perp$. In this case, $P[N/x_0] = N|\overline{a}\langle M_1, \ldots, M_m\rangle$ reduces to $Q[x_{i_k}/x_0][M_k/x_{i_k}] = Q[M_k/x_0]$. Since $M_k \in \mathbf{N}_{i_k}$ and $Q \in \mathbf{N}_{i_k}^\perp$, the latter converges to $\maltese$.

(3) Let $N = \sum_\alpha a(\vec{x}_a).Q_a$ be an l-design in $|\overline{\alpha}^c\langle \mathbf{P}_1^\perp, \ldots, \mathbf{P}_n^\perp\rangle^\perp|_\alpha$. Let $a(\vec{x}_a) \in \alpha$. Since N is orthogonal to $\overline{a}\langle \mathbf{P}_{i_1}^\perp, \ldots, \mathbf{P}_{i_m}^\perp\rangle \subseteq \overline{\alpha}^c\langle \mathbf{P}_1^\perp, \ldots, \mathbf{P}_n^\perp\rangle$, $Q_a[M_1/x_{i_1}, \ldots, M_m/x_{i_m}]$ must converge for all $M_1 \in \mathbf{P}_{i_1}^\perp$, $\ldots$, $M_m \in \mathbf{P}_{i_m}^\perp$. This shows that Q_a belongs to $[\mathbf{P}_{i_1}^\perp/x_{i_1}, \ldots, \mathbf{P}_{i_m}^\perp/x_{i_m}]^\perp$. Hence N belongs to the RHS.

(4) Immediate by definition. $\square$

The internal completeness follows directly from the above lemma.

Theorem 4.14 (Internal completeness)

(1) $|\overline{\alpha}\langle \mathbf{N}_1, \ldots, \mathbf{N}_n\rangle|_h = \bigcup_{a(\vec{x}_a)\in\alpha} \overline{a}\langle \mathbf{N}_{i_1}, \ldots, \mathbf{N}_{i_m}\rangle$.
(2) $|\alpha(\mathbf{P}_1, \ldots, \mathbf{P}_n)|_\alpha = \sum_\alpha a(\vec{x}_a).[\mathbf{P}_{i_1}^\perp/x_{i_1}, \ldots, \mathbf{P}_{i_m}^\perp/x_{i_m}]^\perp$.

Proof (1) The inclusion $\supseteq$ is obvious. The converse inclusion follows from Lemma 4.13 (1) together with $|\overline{\alpha}\langle \mathbf{N}_1, \ldots, \mathbf{N}_n\rangle|_h \subseteq |\alpha^c(\mathbf{N}_1^\perp, \ldots, \mathbf{N}_n^\perp)^\perp|_h$, which is a consequence of (2).

(2) The inclusion $\subseteq$ follows from Lemma 4.13 (3) together with $|\alpha(\mathbf{P}_1, \ldots, \mathbf{P}_n)|_\alpha \subseteq |\overline{\alpha}^c\langle \mathbf{P}_1^\perp, \ldots, \mathbf{P}_n^\perp\rangle^\perp|_\alpha$ which is a consequence of (4).

As to the converse, let $N \in \sum_\alpha a(\vec{x}_a).[\mathbf{P}_{i_1}^\perp/x_{i_1}, \ldots, \mathbf{P}_{i_m}^\perp/x_{i_m}]^\perp$. Then N is of the form $\sum_\alpha a(\vec{x}_a).Q_a$ and $Q_a \in [\mathbf{P}_{i_1}^\perp/x_{i_1}, \ldots, \mathbf{P}_{i_m}^\perp/x_{i_m}]^\perp$ for every $a(\vec{x}_a) \in \alpha$. Let also $P \in \overline{\alpha}^c\langle \mathbf{P}_1^\perp, \ldots, \mathbf{P}_n^\perp\rangle$. Then P is of the form $x_0|\overline{a}\langle M_1, \ldots, M_m\rangle$ for some $a(\vec{x}_a) \in \alpha$, $\vec{x}_a = x_{i_1}, \ldots, x_{i_m}$ and $M_1 \in \mathbf{P}_{i_1}^\perp$, $\ldots$, $M_m \in \mathbf{P}_{i_m}^\perp$. We have $P[N/x_0] \longrightarrow Q_a[M_1/x_{i_1}, \ldots, M_m/x_{i_m}] \Downarrow \maltese$. Hence $N \in (\overline{\alpha}^c\langle \mathbf{P}_1^\perp, \ldots, \mathbf{P}_n^\perp\rangle)^\perp = \alpha(\mathbf{P}_1, \ldots, \mathbf{P}_n)$. $\square$

In particular, we have:

$$
\begin{aligned}
&|\oplus\langle \mathbf{N}, \mathbf{M}\rangle|_h = \iota_1\langle \mathbf{N}\rangle \cup \iota_2\langle \mathbf{M}\rangle, && |\,\&\,(\mathbf{P}, \mathbf{Q})|_\& = \pi_1(x_0).\mathbf{P} + \pi_2(x_0).\mathbf{Q},\\
&|\otimes\langle \mathbf{N}, \mathbf{M}\rangle|_h = \bullet\langle \mathbf{N}, \mathbf{M}\rangle, && |\parr(\mathbf{P}, \mathbf{Q})|_\parr = \parr(x_1, x_2).[\mathbf{P}^\perp/x_1, \mathbf{Q}^\perp/x_2]^\perp.
\end{aligned}
$$

The second one holds because $\pi_1(x_1).[\mathbf{P}^\perp/x_1]^\perp = \pi_1(x_0).\mathbf{P}^{\perp\perp} = \pi_1(x_0).\mathbf{P}$. It is of particular interest. Notice that the LHS is basically an intersection

$\&(\mathbf{P}, \mathbf{Q}) = \iota_1\langle \mathbf{P}^\perp \rangle^\perp \cap \iota_2\langle \mathbf{Q}^\perp \rangle^\perp$ whereas the RHS is a cartesian product. Hence it states that intersection and cartesian product are the same up to incarnation, and is called the *mystery of incarnation* in [9]. Notice also that a unary negative connective behaves like a positive connective, in that the orthogonal operation is completely removable. For instance, for the unary connective $\uparrow = \{\uparrow(x_1)\}$, we have: $|\uparrow(\mathbf{P})|_\uparrow = \uparrow(x_0).\mathbf{P}$.

Corollary 4.15

(1) $|\overline{\alpha}\langle \mathbf{N}_1, \ldots, \mathbf{N}_n \rangle| = \bigcup_{a(\vec{x}_a) \in \alpha} \overline{a}\langle |\mathbf{N}_{i_1}|, \ldots, |\mathbf{N}_{i_m}| \rangle \cup \{\maltese\}$.
(2) $||\overline{\alpha}\langle \mathbf{N}_1, \ldots, \mathbf{N}_n \rangle|| = \bigcup_{a(\vec{x}_a) \in \alpha} \overline{a}\langle ||\mathbf{N}_{i_1}||, \ldots, ||\mathbf{N}_{i_m}|| \rangle$.
(3) $|\alpha(\mathbf{P}_1, \ldots, \mathbf{P}_n)| = \sum_\alpha a(\vec{x}_a).|\,[\mathbf{P}_{i_1}^\perp/x_{i_1}, \ldots, \mathbf{P}_{i_m}^\perp/x_{i_m}]^\perp|$.
(4) $||\alpha(\mathbf{P}_1, \ldots, \mathbf{P}_n)|| = \sum_\alpha a(\vec{x}_a).||\,[\mathbf{P}_{i_1}^\perp/x_{i_1}, \ldots, \mathbf{P}_{i_m}^\perp/x_{i_m}]^\perp||$.

Proof For simplicity, let us assume $n = 1$ and all elements of α are of the form $a(x)$.

(1) If $P \in |\overline{\alpha}\langle \mathbf{N} \rangle|$, then P is normal, and hence is either $\maltese$ or of the form $x|\overline{a}\langle M \rangle$. In the latter case, P belongs to $|\overline{\alpha}\langle \mathbf{N} \rangle|_h$. Hence by internal completeness, $a(x) \in \alpha$ and $M \in \mathbf{N}$. If there is $M' \subsetneq M$ in $\mathbf{N}$, then $x|\overline{a}\langle M' \rangle$ belongs to $\overline{\alpha}\langle \mathbf{N} \rangle$, that contradicts P being material. Hence M is material in $\mathbf{N}$, and we conclude that P belongs to the RHS. The converse direction is easy.

(3) If $N \in |\alpha(\mathbf{P})|$, one can easily observe that N is of the form $\sum_\alpha a(x).P_a$ and belongs to $|\alpha(\mathbf{P})|_\alpha$. Hence by internal completeness, $P_a \in [\mathbf{P}^\perp/x]^\perp$ for every $a(x) \in \alpha$. If there is $P'_a \subsetneq P_a$ in $[\mathbf{P}^\perp/x]^\perp$, then $\sum_\alpha a(x).P'_a$ belongs to $\alpha(\mathbf{P})$, that contradicts N being material. Hence P_a is material in $[\mathbf{P}^\perp/x]^\perp$, and we conclude that N belongs to the RHS.

(2) and (4) easily follow from (1) and (3), respectively. □

4.5 Defining data sets by construction

As we have noted at the beginning of subsection 4.3, two approaches for defining a language (by interaction and by construction) generalize to the setting of defining a set of designs: by orthogonality and by logical connectives. While the first approach necessarily leads to behaviours which are biorthogonal closed, the second abhors biorthogonals. Hence for the two approaches to reside in harmony, sets of designs in question must be biorthogonal closed, and yet the biorthogonals must be removable. That is exactly what the internal completeness theorem achieves. Let us now see its effect in the concrete setting of data designs.

First, notice that internal completeness of logical connectives (Corollary 4.15) yields the following:

Proposition 4.16 *For any n-ary logical connective α and atomic negative behaviours $\mathbf{N}_1, \ldots, \mathbf{N}_n$, we have:*

$$|| \uparrow \overline{\alpha}\langle \mathbf{N}_1, \ldots, \mathbf{N}_n \rangle || = \bigcup_{a(\vec{x}_a) \in \alpha} \uparrow \overline{a}\langle\, ||\mathbf{N}_{i_1}||, \ldots,\ ||\mathbf{N}_{i_m}||\, \rangle,$$

where indices $i_1, \ldots, i_m$ depends on $a(\vec{x}_a) \in \alpha$ as before.

This allows us to construct various behaviours by means of logical connectives.

Empty set. Consider the empty logical connective $\emptyset$. We have:

$$|| \uparrow \overline{\emptyset} || = \emptyset.$$

Singleton. Denote the 0-ary logical connective $\{\mathsf{nil}\}$ by Nil. We then have:

$$|| \uparrow \overline{\mathsf{Nil}} || = \{\uparrow \overline{\mathsf{nil}}\}.$$

Prefixed union. Let $\beta = \{a(x_1), b(x_2)\}$. Then,

$$|| \uparrow \overline{\beta}\langle \mathbf{M}, \mathbf{N} \rangle || = \uparrow \overline{a}\langle ||\mathbf{M}|| \rangle \cup \uparrow \overline{b}\langle ||\mathbf{N}|| \rangle.$$

The above constructions are all based on logical connectives, and apply to arbitrary behaviours. On the other hand, the constructions below are specific to those representing data sets. First, let us observe two simple facts:

Lemma 4.17 *Let $\mathbf{D}$ be a set of data designs. Then we have $||\mathbf{D}^{\perp\perp}|| = \mathbf{D}$.*

Proof Let $N \in ||\mathbf{D}^{\perp\perp}||$. Since the counter design $c(\mathbf{D})$ (Definition 3.17) belongs to $\mathbf{D}^{\perp}$, we have $N \perp c(\mathbf{D})$. Hence by Theorem 3.20, $d \sqsubseteq N$ for some $d \in \mathbf{D}$. By materiality of N, we have $N = d$. $\square$

Lemma 4.18 *Let $\mathbf{N}$ be a set of atomic negative l-designs, and $\mathbf{M}$ be a set of negative l-designs with at most one free variable x. We then have*

$$(\mathbf{M}[\mathbf{N}/x])^{\perp} = (\mathbf{M}^{\perp\perp}[\mathbf{N}^{\perp\perp}/x])^{\perp},$$

where $\mathbf{M}[\mathbf{N}/x] = \{M[N/x] : M \in \mathbf{M}, N \in \mathbf{N}\}$.

Proof Observe that for any anti-design $[G]$ against negatives, we have

(*) $\forall M \in \mathbf{M}.\ [G] \perp M \iff [G] \in \mathbf{M}^{\perp} \iff \forall M \in \mathbf{M}^{\perp\perp}.\ [G] \perp M.$

Similarly for $\mathbf{N}$. Now, $P \in (\mathbf{M}[\mathbf{N}/x])^{\perp}$ iff $\forall M \in \mathbf{M}.\forall N \in \mathbf{N}.P[M[N/x]/x_0] \Downarrow$. Moreover, $P[M[N/x]/x_0] \Downarrow$ is equivalent to both $P[M/x_0]\perp[N/x]$ (i.e. $P[M/x_0][x_0/x]\perp N$) and $[P, N/x]\perp M$. Hence by using (*) twice, we derive the desired equality. $\square$

Union. Given negative behaviours $\mathbf{M}$ and $\mathbf{N}$, one can form another behaviour $(\mathbf{M}\cup\mathbf{N})^{\perp\perp}$. When $\mathbf{M} = \mathbf{D}^{\perp\perp}$ and $\mathbf{N} = \mathbf{E}^{\perp\perp}$ for some sets $\mathbf{D}$, $\mathbf{E}$ of data designs, this indeed works as the union operator by Lemma 4.17:

$$||(\mathbf{M}\cup\mathbf{N})^{\perp\perp}|| = ||(\mathbf{D}^{\perp\perp}\cup\mathbf{E}^{\perp\perp})^{\perp\perp}|| = ||(\mathbf{D}\cup\mathbf{E})^{\perp\perp}|| = \mathbf{D}\cup\mathbf{E} = ||\mathbf{M}||\cup||\mathbf{N}||.$$

Composition. Given $\mathbf{M}$, $\mathbf{N}$ as in Lemma 4.18, we may form another behaviour $(\mathbf{M}[\mathbf{N}/x])^{\perp\perp}$. When $\mathbf{M} = \mathbf{D}_x^{\perp\perp}$ and $\mathbf{N} = \mathbf{E}^{\perp\perp}$, where $\mathbf{E}$ is a set of data designs and $\mathbf{D}_x$ consists of l-designs which are like data designs but are allowed to have at most one occurrence of identity x, we have:

$$||(\mathbf{M}[\mathbf{N}/x])^{\perp\perp}|| = ||(\mathbf{D}_x[\mathbf{E}/x])^{\perp\perp}|| = \mathbf{D}_x[\mathbf{E}/x] = ||\mathbf{M}||\,[\,||\mathbf{N}||/x\,].$$

Iteration. Given $\mathbf{M} = \mathbf{D}_x^{\perp\perp}$ as above, we may define $\mathbf{M}^0 = \uparrow \overline{\mathsf{Nil}}$, $\mathbf{M}^{n+1} = (\mathbf{M}[\mathbf{M}^n/x])^{\perp\perp}$, $\mathbf{M}^* = (\bigcup_n \mathbf{M}^n)^{\perp\perp}$. Then from what precede, we derive:

$$||\mathbf{M}^*|| = \bigcup_n ||\mathbf{M}||^n.$$

We thus have ludics analogues of language operators defiining regular languages. However, it should be noted that while prefixed union properly works for arbitrary behaviours by internal completeness, union, composition and iteration only work for behaviours arising from data designs, since the latter rely on the strong separation property (Theorem 3.20).

5 Conclusion

We have reformulated ludics from a computational point of view. Our syntax is designed for representing various algorithms in it. For that purpose, having explicit cuts inside c-designs is of vital importance. Another important issue is finite generation of infinite c-designs. The significance of cuts and finite generation is well summarized by the three characterization results (Theorem 4.10):

(1) Arbitrary l-designs may capture arbitrary sets of finite data designs.
(2) Finitely generated l-designs exactly capture the recursively enumerable languages (when restricted to acceptance of languages).

(3) Finitely generated cut-free l-designs (i.e., finitely generated standard c-designs) exactly capture the regular languages.

To prove these results, we have made essential uses of the fundamental properties of ludics, such as associativity, separation, pull-back, incarnation, and internal completeness. Our development illustrates how useful these apparently abstract properties are for practical purposes.

We have also explained how two approaches for defining languages in automata theory generalize to the ludics setting. The interaction approach leads to the notion of orthogonality, while the construction approach leads to logical connectives and other constructions. These two approaches are compatible thanks to the internal completeness theorem and the strong separation property.

Our subsequent work will discuss:

Syntactic types. In subsection 4.5, we have launched construction of various behaviours by logical connectives (and other means). This approach can be most effectively pursued by introducing syntactic types, which are analogous to (regular) expressions in automata theory. Then the issue of internal completeness, together with its applications, naturally carries over to the issue of *full completeness*, a full correspondence between types and behaviours.

Focalization. *Space compression theorem*, one of the most fundamental results in complexity theory, is based on a very simple idea of compressing data by using more symbols: a typical example is the transformation of natural numbers in base 2 to those in base 4:

$$(0110)_2 \longmapsto (12)_4.$$

In terms of data designs, this corresponds to the following map:

$$\uparrow\overline{0}\langle\uparrow\overline{1}\langle\uparrow\overline{1}\langle\uparrow\overline{0}\langle\uparrow\overline{\mathsf{nil}}\rangle\rangle\rangle\rangle \ \longmapsto \uparrow\overline{1}\langle\uparrow\overline{2}\langle\uparrow\overline{\mathsf{nil}}\rangle\rangle.$$

Interestingly, this map can be derived from a general principle of *focalization*:

$$\overline{\alpha}\langle\uparrow\overline{\beta}\langle\mathbf{N}\rangle\rangle \cong \overline{\alpha\beta}\langle\mathbf{N}\rangle,$$

which states that two consecutive logical connectives α, β of the same polarity (here separated by $\uparrow$) can be combined into one $\alpha\beta$. In our subsequent work, we plan to prove a general form of focalization in ludics, and derive space compression from that, aiming at a logical account to the latter computational phenomenon.

Extensions of ludics. Recently Basaldella and Faggian have extended designs and behaviours to non-linear settings. Non-linear designs are important in various ways. In particular, what we have in mind is to use them to give a logical account to space sensitive composition (eg. composition of logspace functions).

Another interesting direction would be to study parallel designs like L-nets [6], from the viewpoint of automata theory, since they are suitable for expressing acceptance of tree languages (see Remark 2.24).

Acknowledgments. We would like to thank Claudia Faggian and the anonymous referees for a lot of useful suggestions.

References

[1] P.-L. Curien. Abstract böhm trees. *Mathematical Structures in Computer Science*, 8(6):559–591, 1998.

[2] P.-L. Curien. Introduction to linear logic and ludics, part II. *CoRR*, abs/cs/0501039, 2005.

[3] D. Du and K.-I. Ko. *Theory of Computational Complexity*. Wiley-Interscience, 2000.

[4] C. Faggian. Interactive observability in ludics: The geometry of tests. *Theor. Comput. Sci.*, 350(2-3):213–233, 2006.

[5] C. Faggian and M. Hyland. Designs, disputes and strategies. In *CSL'02*, pages 442–457, 2002.

[6] C. Faggian and F. Maurel. Ludics nets, a game model of concurrent interaction. In *LICS'05*, pages 376–385, 2005.

[7] C. Faggian and M. Piccolo. Ludics is a model for the finitary linear pi-calculus. In *TLCA'07*, pages 148–162, 2007.

[8] J.-Y. Girard. Linear logic: Its syntax and semantics. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 1–42. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.

[9] J.-Y. Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3):301–506, 2001.

[10] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1988.

[11] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[12] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–259, 1997.

[13] F. Joachimski. Confluence of the coinductive [lambda]-calculus. *Theor. Comput. Sci.*, 311(1-3):105–119, 2004.

[14] J.-L. Krivine. Typed lambda-calculus in classical Zermelo-Fraenkel set theory. *Arch. Math. Log.*, 40(3):189–205, 2001.

[15] J.-L. Krivine. A call-by-name lambda calculus machine. Available at http://www.pps.jussieu.fr/$\sim$ krivine, 2006.

[16] O. Laurent. Polarized games. *Ann. Pure Appl. Logic*, 130(1-3):79–123, 2004.

[17] P.-A. Melliès and J. Vouillon. Recursive polymorphic types and parametricity in an operational framework. In *LICS'05*, pages 82–91, 2005.