

Computability structures, simulations and realizability

John Longley

May 12, 2011

Abstract

We generalize the standard construction of realizability models (specifically, of categories of assemblies) to a very wide class of *computability structures*, broad enough to embrace models of computation such as labelled transition systems and process algebras. We also discuss a general notion of *simulation* between such computability structures, and show that such simulations correspond precisely to certain functors between the realizability models. Furthermore, we show that our class of computability structures has good closure properties — in particular, it is ‘cartesian closed’ in a slightly relaxed sense. We also investigate some important subclasses of computability structures and of simulations between them. We suggest that our 2-category of computability structures and simulations may offer a framework for a general investigation of questions of computational power, abstraction and simulability for a wide range of computation models from across computer science.

1 Introduction

1.1 Background

The main purpose of this paper is to present a rather broad generalization of the standard construction of realizability models — one that is general enough to apply to a wide range of models of computation from many areas of computer science.

In the standard account of realizability models (as presented e.g. in [22]), one begins with a *partial combinatory algebra* (or PCA) A — a structure which one may loosely regard as a kind of ‘abstract machine’ or ‘untyped model of computation’. From this, one builds a category such as the *category of assemblies* $\mathbf{Asm}(A)$, whose objects can be informally thought of as ‘datatypes’ that can be represented or implemented on this machine, and whose morphisms may be thought of as ‘ A -computable maps’ between such datatypes. (We shall briefly review the construction of $\mathbf{Asm}(A)$ in Section 2 below.) Such categories possess a very rich mathematical structure and have proved interesting from several points of view: for instance, as categorical models corresponding to Kleene-style realizability interpretations of intuitionistic logic [8]; as models for powerful polymorphic type systems [17]; or as universes within which certain ‘sets’ carry an intrinsic domain-like structure [9, 20].

In [12], a notion of morphism between PCAs was introduced which interacts well with the \mathbf{Asm} construction. Informally, an *applicative morphism* $A \rightarrow B$ is a way of simulating or implementing the abstract machine A on the machine B ; in many cases it makes sense

to think of A here as modelling computation at a ‘more abstract’ level than B . Such a simulation induces a functor $\mathbf{Asm}(A) \rightarrow \mathbf{Asm}(B)$; moreover, one can precisely characterize those functors that arise in this way up to natural isomorphism. In this way, we obtain a tight correspondence between the (2-)category of PCAs and applicative morphisms (which we call \mathcal{PCA}) and a certain (2-)category of categories. (Again, all this material will be reviewed in Section 2.)

Our 2-category \mathcal{PCA} gives rise to a notion of *equivalence* between PCAs. Informally, two PCAs A, B are equivalent if there are simulations $A \rightarrow B$ and $B \rightarrow A$ which are ‘mutually inverse up to computable translation’. This notion highlights some interesting differences in ‘computational strength’ between well-known models of computation: for example, the PCA consisting of the natural numbers with Kleene application turns out to be *inequivalent* to the PCA of untyped λ -terms modulo β -equality, even though these models have the same computational power as measured by the functions $\mathbb{N} \rightarrow \mathbb{N}$ that they can compute.¹

A consequence of the above theory is that A and B are equivalent if and only if $\mathbf{Asm}(A)$ and $\mathbf{Asm}(B)$ are equivalent as categories. This suggests that we may in some sense view equivalent PCAs as just alternative presentations of the same underlying ‘notion of computability’, and that the category of assemblies gives us a kind of presentation-invariant embodiment of what it is they have in common. (See however Section 6.1 below for an important qualification on this idea.) Indeed, if one’s main interest is in the PCAs themselves and their interrelationships, one can see the significance of the above correspondence as being partly that it offers some sort of vindication for the definition of our 2-category \mathcal{PCA} (and for the resulting notion of PCA equivalence).

All this is a pleasing enough story as far as it goes, and the correspondence theorem we have mentioned is mathematically satisfying. However, this story might be felt to be of limited interest on at least two accounts:

- The vast majority of ‘models of computation’ studied in computer science do not naturally take the form of PCAs. However, the general concepts of ‘computational power’, ‘simulation’ and ‘levels of abstraction’ would in principle seem to be of a very general interest, so it is natural to wonder whether our mathematical framework for discussing such concepts could be broadened to encompass other models of computation, such as labelled transition systems or process calculi.
- The 2-category \mathcal{PCA} turns out to be relatively poor in 2-categorical structure. In the interests of coming up with a mathematically fruitful theory, it is natural to wonder whether we can do better.

As regards the first of these points, some progress was made in [13, 14, 11], where it was shown that the basic theory can be straightforwardly generalized to a setting of *typed PCAs*, of which ordinary (untyped) PCAs form a special case. This greatly extends the scope of the theory: for instance, the typed PCA framework embraces all the cartesian closed categories used in denotational semantics, as well as a wealth of syntactic models for languages such as PCF and extensions thereof. This version of the theory was presented

¹This example is explained in detail in [12, Chapter 3]. Briefly, each of these models may be simulated in the other, but the problem is that the simulations cannot be ‘essentially mutually inverse’ in the required sense, owing to the fact that the λ -calculus is in some way ‘more abstract’ than the Kleene model.

more fully in [16], where the typed PCA framework played a crucial role in an investigation of higher order computability notions. A further generalization to ‘linear’ (in contrast to ‘intuitionistic’) structures was presented in [7].

Even so, the typed PCA framework comes nowhere near to embracing the enormous variety of ‘models of computation’ currently studied in computer science as a whole. Moreover, from the point of view of categorical structure, the 2-category of typed PCAs is scarcely better than our original 2-category of PCAs. We may therefore still ask whether it is possible to do better on both these scores.

1.2 Content of the paper

Our purpose in this paper is to offer a significantly more general framework for the study of models of computation and the ‘notions of computability’ they embody. Specifically, we shall define a rather broad 2-category *CSTRUCT* of *computability structures* and *simulations* between them, much broader than the 2-category of typed PCAs, in such a way that:

- a version of the **Asm** construction still makes sense, and an appropriate generalization of the correspondence theorem goes through;
- a far wider range of models from across computer science are admitted — for example, labelled transition systems and process calculi;
- the class of computability structures has better closure properties than that of (un-typed or typed) PCAs.

Our work may be motivated by either mathematical or computer science considerations. From a purely mathematical point of view, it can be seen as a deeper investigation the correspondence theorem first established in the PCA setting: just how far does this phenomenon extend, and what is the most general setting in which it naturally works? Moreover, the identification of a class of models with richer closure properties is of mathematical interest in itself, and is perhaps an indication that we are ploughing fertile ground. From a computer science point of view, our hope is to offer a general framework for a study of ‘computability’ and ‘simulations’ that does justice to the diversity of computation models currently in use. Much of the richness of computer science, after all, comes from the use of a wide variety of modelling styles to study computational systems at many different levels of description or abstraction. A general framework for talking about these modelling styles and their interrelationships therefore holds out the hope of bringing some level of unification to hitherto disparate parts of the subject. Already, the typed PCA framework has proved itself useful for bringing results from diverse sources together in a uniform framework where they may be readily compared and combined (see e.g. [16]); and we are hopeful that this may be true to an even greater extent for the much broader framework of computability structures.

Perhaps the key idea behind our generalization is a switch from a ‘higher order’ to a ‘first order’ style of modelling. Typed PCAs naturally model ‘higher order’ flavours of computation, in the sense that a ‘computable operation’ from a type σ to a type τ is a value of some type $(\sigma \rightarrow \tau)$ which may itself serve as input to another computable operation. In the first order style of modelling, data values are rigidly separated from the computable

operations that act on them, and only later do we show how the higher order situation may be conveniently recovered as a special case of this. One of the main goals of this paper is to demonstrate how surprisingly much of the basic machinery of realizability models can be smoothly adapted to this ‘flattened’ first order setting.

Two major omissions should be noted at the outset. First, our focus in this paper is mostly on setting up the general framework, and there will be relatively little investigation into particular models. Although in Section 3.1 we shall briefly indicate some of the main sources of examples we have in mind, it would require a far deeper study of specific models to substantiate our claims regarding the unifying potential of our framework (for instance, it would be very interesting to see how the landscape of process algebras appears from the point of view of our framework). Secondly, whilst we shall pay some attention to the *categorical* structure of $\mathbf{Asm}(\mathbf{C})$ for various kinds of model \mathbf{C} , we shall barely touch on the kinds of *logical* constructs that such categories allow us to interpret. Insofar as the interpretation of logic is a key motivation for the study of realizability in the first place, this might be seen as a serious deficiency. On the other hand, readers experienced in categorical logic will have little difficulty in seeing broadly what kinds of logical structure our categories support.

The main body of the paper is structured as follows. In Section 2 we give a brief technical summary of the relevant parts of the existing theory of untyped and typed PCAs. In Section 3 we define our main category of interest: the (2-)category $\mathit{CSTRUCT}$ of *computability structures* and *simulations*. We also show how any computability structure \mathbf{C} gives rise to a ‘category of assemblies’ $\mathbf{Asm}(\mathbf{C})$, and establish a tight correspondence between simulations $\mathbf{C} \rightarrow \mathbf{D}$ and certain functors $\mathbf{Asm}(\mathbf{C}) \rightarrow \mathbf{Asm}(\mathbf{D})$, generalizes the existing correspondence theorem for PCAs. In Section 4 we investigate the closure properties of $\mathit{CSTRUCT}$, showing in particular that it is ‘almost cartesian closed’, in a sense that is good enough to ensure that for any structures \mathbf{C} and \mathbf{D} , the structure ‘ $\mathbf{D}^{\mathbf{C}}$ ’ is uniquely determined up to equivalence. In Section 5 we mention some possible variations on our theory and some interesting sub-categories of $\mathit{CSTRUCT}$; in particular we show how the theory of typed PCAs fits into this more general framework. We end in Section 6 with some general discussion and suggestions for further work.

1.3 Related work

Our work overlaps significantly with recent and essentially independent work by Cockett and Hofstra [4]. In both their work and ours, the key idea is to generalize the definition of PCA to something with a typed, first order flavour, and to show how a general notion of ‘simulation’ may be defined in this setting. Another point in common is the idea that models of computation give rise to categories of certain kinds, in such a way that simulation equivalence corresponds precisely to some well-behaved kind of categorical equivalence. In both cases, then, the basic philosophy is that the category serves as a kind of presentation-invariant embodiment of the underlying ‘notion of computability’.

Various differences between [4] and our work should also be noted. Firstly, there is a general difference in spirit: whereas Cockett and Hofstra adopt a thoroughgoing ‘categorical’ approach that provides a detailed and precise axiomatization of the mathematical structure that the theory presupposes, we have opted for a relatively simple-minded ‘set theoretic’ treatment of models of computation as they concretely appear to the working

computer scientist. Secondly, at a technical level, there are various differences in the precise level of generality in which the phenomena in question are studied, even once the categorical framework of [4] has been specialized to a familiar set-theoretic setting. Some of these differences will be pointed out at the relevant places in our technical development (see Section 3.1). Nevertheless, there is a very significant area of overlap in which we are talking about essentially the same things — in particular, it is striking that our notion of ‘equivalence of models’ turns out to coincide exactly (for models within the overlap) with the *lax simulation equivalence* of [4]. Thirdly, as regards the categories derived from the computation models, the focus of the two approaches is somewhat different. Whereas our aim is to describe concretely a construction that directly generalizes that of the ‘classical’ category of assemblies, the aim in [4] (see also [3]) is to give a categorical analysis of this construction in terms of several smaller steps. Indeed, most of the focus in [3, 4] is not on the classical category of assemblies itself, but on an intermediate structure known as a *Turing category*. One of the main results of [4] is a correspondence theorem for Turing categories, closely analogous to ours for categories of assemblies. We will resume discussion of this in Section 6.1 below.

Aside from these differences, the following new technical contributions are specific to the present work. First, we identify a precise categorical characterization of those functors between assembly categories that arise from simulations between computability structures; this enables us to state and prove our version of the correspondence theorem. The key notion here is that of a *quasi-regular* functor — a generalization of the notion of regular functor to a setting in which finite products are not assumed. Secondly, our identification of the ‘almost cartesian closed’ structure in our 2-category of models is novel. Thirdly, there is an explicit emphasis in our work on the generalization to *non-deterministic* models of computation such as those arising from concurrency theory, and in some respects our definitions are tuned so as to admit a large class of examples from process algebra and related areas (see Example 3.3(iii) and the remark following).

Also similar in spirit to our work is the general framework for realizability developed by Hofstra in [6]. Here one starts with a broad 2-category of *basic combinatorial objects*, whose definition resembles that of our computability structures in some respects. On one axis, Hofstra’s framework is more restricted than others: since his main interest is in what one needs in order to build a realizability *topos* (via a tripos), he restricts attention to untyped structures rather than general typed ones. Along another axis, however, his framework is broader in that it embraces the notion of *ordered PCAs*, a generalization of ordinary PCAs introduced in [21]. As shown in [6], this extra generality allows one to unify ‘computational’ examples from with geometric or ‘localic’ ones. Since our present focus is on a framework for models of computation, these geometric examples are perhaps of lesser importance, and we have chosen to work in the slightly simpler ‘non-ordered’ setting, at the cost of some additional generality. Nevertheless, we expect that most if not all of our theory extends readily to the ordered setting, and this might be a fruitful area for further investigation.

Finally, we mention the work of Abramsky on ‘process realizability’ [1], a notable attempt to broaden the scope of realizability to embrace concurrent and non-deterministic flavours of computation as well as the usual functional or applicative kinds. In [1], the ideas are developed in the setting of (a mild extension of) Milner’s CCS. In place of a single ‘application’ operation as in the theory of PCAs, one considers both a ‘left’ and a ‘right’ application, dual to each other, which allows the same process term to act as a ‘function’

in two entirely symmetrical ways. When transposed to our present setting, an effect of the flattening from ‘higher order’ to ‘first order’ is that both these kinds of application are subsumed as instances of the more general concept of computable operation. However, the realizability construction Abramsky considers is interestingly different from the standard one, and exploits the duality inherent in his setup to give a model of Classical Linear Logic. It would be interesting to know how widely a version of this ‘self-dual’ realizability construction can be applied in our general setting.

Acknowledgements

I am very grateful to Robin Cockett and Pieter Hofstra for the invitation to present this material at the PCARC workshop, for illuminating technical discussions, and for the encouragement to write this paper. I am also grateful to the other PCARC participants for their valuable feedback.

I have benefited from the use of Paul Taylor’s LaTeX package for category theory diagrams in the production of this paper.

2 Untyped and typed PCAs

We start with a brief review of the existing definitions and results for untyped and typed PCAs; for fuller details see [12] or [22]. Later we shall recover this material as a special case of our more general framework.

If e, e' are mathematical expressions, we write $e \downarrow$ to mean “ e is defined”; $e \simeq e'$ to mean “if either e or e' is defined then so is the other and their values are equal”; and $e \succeq e'$ to mean “if e' is defined then so is e and their values are equal”.

2.1 Untyped PCAs

The following definition (with minor differences) appeared in [5]; it generalizes the notion of a total combinatory algebra introduced in [19].

Definition 2.1 *An (untyped) partial combinatory algebra, or PCA, is a set A equipped with a (left-associative) partial binary operation $\cdot : A \times A \rightarrow A$ and containing elements k, s such that*

$$k \cdot x \cdot y = x \quad s \cdot x \cdot y \downarrow \quad s \cdot x \cdot y \cdot z \succeq x \cdot z \cdot (y \cdot z)$$

Here and throughout the paper, we adopt that convention that variables in displayed formulae that are not explicitly introduced elsewhere (in this case, x, y, z) are regarded as implicitly universally quantified at the start of the displayed formula.

Variations on the above definition are possible. For instance, the definition is more usually given with ‘ \simeq ’ in place of ‘ \succeq ’, but the theory we shall present works naturally with our slightly more general definition. Furthermore, the condition $s \cdot x \cdot y \downarrow$ is unnecessary for many purposes (see [22, Theorem 1.2.3]), although it renders the development of our theory somewhat smoother.

The following definition first appeared in [12]:

Definition 2.2 Let A, B be PCAs.

(i) An applicative morphism $\gamma : A \multimap B$ is a total relation γ from A to B such that for some $r \in B$ we have

$$\gamma(a, b) \wedge \gamma(a', b') \wedge a \cdot a' \downarrow \Rightarrow \gamma(a \cdot a', r \cdot b \cdot b')$$

(ii) Given $\gamma, \delta : A \multimap B$, we write $\gamma \preceq \delta$, and say there is an applicative transformation from γ to δ , if for some $t \in B$ we have

$$\gamma(a, b) \Rightarrow \delta(a, t \cdot b)$$

An applicative morphism $A \multimap B$ can be seen as a way of simulating A in B ; a convenient slogan is “application in A is effective in B ”. The condition $\gamma \preceq \delta$ says that γ -representations of elements of A may be transformed into δ -representations (effectively in B). It is easy to check that PCAs, applicative morphisms and applicative transformations constitute a preorder-enriched category, which we call \mathcal{PCA} .

We now show how to construct a realizability model over a PCA.

Definition 2.3 For any PCA A , the category of assemblies $\mathbf{Asm}(A)$ is defined as follows.

- Objects are pairs (X, \Vdash) where X is a set and $\Vdash \subseteq A \times X$ is a relation such that for every $x \in X$ there is some $a \Vdash x$.
- Morphisms $(X, \Vdash) \rightarrow (Y, \Vdash')$ are set-theoretic functions $f : X \rightarrow Y$ such that for some $r \in A$ we have

$$a \Vdash x \Rightarrow r \cdot a \Vdash' f(x)$$

Identities and composition are as for ordinary functions.

The category $\mathbf{Asm}(A)$ is equipped with an evident forgetful functor $\Gamma_A : \mathbf{Asm}(A) \rightarrow \mathbf{Set}$, and also a functor $\nabla_A : \mathbf{Set} \rightarrow \mathbf{Asm}(A)$ defined as follows: $\nabla_A(X) = (X, \Vdash)$ where $a \Vdash x$ for all a and x , and $\nabla_A(f) = f$.

It turns out that $\mathbf{Asm}(A)$ is among other things a *regular* category: that is, it has finite limits and coequalizers of kernel-pairs, and the pullback of a regular epi along any morphism is a regular epi. (An in-depth understanding of this definition is not needed for this paper.) Furthermore, both Γ_A and ∇_A are *regular functors* (also known as exact functors): that is to say, they preserve finite limits and coequalizers of kernel-pairs. Note also that Γ_A is left adjoint to ∇_A and that $\Gamma_A \circ \nabla_A = \text{id}$.

An applicative morphism $\gamma : A \multimap B$ induces a functor $\mathbf{Asm}(\gamma) : \mathbf{Asm}(A) \rightarrow \mathbf{Asm}(B)$ in the following way: $\mathbf{Asm}(\gamma)(X, \Vdash) = (X, \Vdash')$ where $b \Vdash' x$ iff there exists $a \Vdash x$ with $\gamma(a, b)$; and $\mathbf{Asm}(\gamma)(f) = f$. It is easy to show that $\mathbf{Asm}(\gamma)$ is a regular functor, and that $\Gamma_B \circ \mathbf{Asm}(\gamma) \cong \Gamma_A$ and $\mathbf{Asm}(\gamma) \circ \nabla_A \cong \nabla_B$. Moreover, an applicative transformation $\gamma \preceq \delta$ induces a natural transformation $\mathbf{Asm}(\gamma) \rightarrow \mathbf{Asm}(\delta)$.

Abstracting from this situation, we arrive at the following definition (we write $*$ to denote horizontal composition):

Definition 2.4 (i) A $\Gamma\nabla$ -regular category is any regular category \mathcal{C} equipped with regular faithful functors $\Gamma : \mathcal{C} \rightarrow \mathbf{Set}$ and $\nabla : \mathbf{Set} \rightarrow \mathcal{C}$ such that $\Gamma \dashv \nabla$ and $\Gamma \circ \nabla \cong \text{id}$.

(ii) If \mathcal{C}, \mathcal{D} are $\Gamma\nabla$ -regular categories, a $\Gamma\nabla$ -regular functor $F : \mathcal{C} \rightarrow \mathcal{D}$ is a regular functor such that $\Gamma_{\mathcal{D}} \circ F \cong \Gamma_{\mathcal{C}}$ and $F \circ \nabla_{\mathcal{C}} \cong \nabla_{\mathcal{D}}$. (These isomorphisms are unique when they exist.)

(iii) If $F, G : \mathcal{C} \rightarrow \mathcal{D}$ are $\Gamma\nabla$ -regular functors, a $\Gamma\nabla$ -natural transformation $\eta : F \rightarrow G$ is a natural transformation such that $\Gamma_{\mathcal{D}} * \eta = \text{id}_{\Gamma_{\mathcal{C}}}$ and $\eta * \nabla_{\mathcal{C}} = \text{id}_{\nabla_{\mathcal{D}}}$. (Both these conditions are automatic in the case $\mathcal{C} = \mathbf{Asm}(A)$, $\mathcal{D} = \mathbf{Asm}(B)$.)

(iv) We write $\Gamma\nabla\mathcal{REG}$ for the (large) 2-category of $\Gamma\nabla$ -regular categories, $\Gamma\nabla$ -regular functors and natural transformations between them.

Actually, the ∇ condition in part (ii) is not strictly needed, as a simple cardinality argument (appearing as Lemma 1.6.3 in [22]) shows that for any functor $F : \mathbf{Asm}(A) \rightarrow \mathbf{Asm}(B)$, the condition $\Gamma_B \circ F \cong \Gamma_A$ automatically implies $F \circ \nabla_A \cong \nabla_B$.

The tight correspondence between applicative morphisms and regular functors is encapsulated in the following theorem from [12] (see also [22]).

Theorem 2.5 *The \mathbf{Asm} construction constitutes a 2-functor $\mathcal{PCA} \rightarrow \Gamma\nabla\mathcal{REG}$ that is locally an equivalence: that is, for any A, B the ordinary functor $\mathbf{Asm} : \mathcal{PCA}[A, B] \rightarrow \Gamma\nabla\mathcal{REG}[\mathbf{Asm}(A), \mathbf{Asm}(B)]$ is part of an equivalence of preorders.*

Corollary 2.6 *$\mathbf{Asm}(A)$ and $\mathbf{Asm}(B)$ are equivalent as categories iff A, B are applicatively equivalent (that is, $A \simeq B$ in the 2-category \mathcal{PCA}).*

The corollary follows because any equivalence of categories $\mathbf{Asm}(A) \simeq \mathbf{Asm}(B)$ may be easily seen to respect the Γ and ∇ functors.

2.2 Typed PCAs

We now briefly sketch the generalization of the above theory to *typed PCAs* as in [13, 14]. (Note that typed PCAs were called ‘partial combinatory type structures’ in [14].) The essential difference from untyped PCAs is that in place of a single carrier set A , we now allow a whole family of carrier sets indexed by *types*.

Definition 2.7 (i) *A type world is simply an inhabited set T of type names (henceforth types) endowed with right-associative binary operations $*$ and \Rightarrow .*

(ii) *A typed PCA (or TPCA) over a type world T is a family of inhabited sets $(A_{\sigma} \mid \sigma \in T)$ equipped with partial ‘application’ operations $\cdot_{\sigma\tau} : A_{\sigma\Rightarrow\tau} \times A_{\sigma} \rightarrow A_{\tau}$ for each $\sigma, \tau \in T$, such that for every $\rho, \sigma, \tau \in T$ there exist elements*

$$\begin{aligned} k_{\sigma\tau} &\in A_{\sigma\Rightarrow\tau\Rightarrow\sigma} & s_{\rho\sigma\tau} &\in A_{(\rho\Rightarrow\sigma\Rightarrow\tau)\Rightarrow(\rho\Rightarrow\sigma)\Rightarrow\rho\Rightarrow\tau} \\ \text{pair}_{\sigma\tau} &\in A_{\sigma\Rightarrow\tau\Rightarrow(\sigma*\tau)} & \text{fst}_{\sigma\tau} &\in A_{(\sigma*\tau)\Rightarrow\sigma} & \text{snd}_{\sigma\tau} &\in A_{(\sigma*\tau)\Rightarrow\tau} \end{aligned}$$

satisfying the following for all appropriately typed x, y, z :

$$\begin{aligned} k \cdot x \cdot y &= x & s \cdot x \cdot y &\downarrow & s \cdot x \cdot y \cdot z &\succeq x \cdot z \cdot (y \cdot z) \\ \text{fst} \cdot (\text{pair} \cdot x \cdot y) &= x & \text{snd} \cdot (\text{pair} \cdot x \cdot y) &= y \end{aligned}$$

Note that TPCAs are inherently higher order: an ‘operation’ that maps values of type σ to values of type τ is itself a value of type $\sigma \Rightarrow \tau$. In the study of higher order computability we are often interested in the type world freely generated from some set of basic type names by means of $*$ and \Rightarrow ; a TPCA over such a type world may be called a *simply typed* PCA. Note also that an untyped PCA is just a TPCA over the singleton type world $\mathbf{1}$.

The remaining definitions from Section 2.1 can be straightforwardly adapted to the TPCA setting by means of suitable type decorations. For instance, if A, B are TPCAs over type worlds T, U respectively, an applicative morphism $\gamma : A \multimap B$ consists of a function $\gamma : T \rightarrow U$ together with a family of total relations γ_σ from A_σ to $B_{\gamma\sigma}$, such that each application operation $\cdot_{\sigma\tau}$ in A is ‘tracked’ by some $r \in B_{\gamma(\sigma\Rightarrow\tau)\Rightarrow\gamma\sigma\Rightarrow\gamma\tau}$ in the sense of Definition 2.2(i). With the evident definition of an applicative transformation $\gamma \preceq \delta$, this yields a preorder-enriched category TPCA of which PCA is a full sub-2-category.

An assembly over a TPCA A consists of a triple (X, σ, \Vdash) where X is a set, σ is a type, and $\Vdash \subseteq A_\sigma \times X$ is a relation such that for every $x \in X$ there is some $a \Vdash X$. A morphism $(X, \sigma, \Vdash) \rightarrow (Y, \tau, \Vdash')$ of such assemblies is just a function $X \rightarrow Y$ that is tracked by some $r \in A_{\sigma\Rightarrow\tau}$ in the obvious way. Once again, the category $\mathbf{Asm}(A)$ of assemblies over a TPCA A turns out to be a $\Gamma\nabla$ -regular category, and the \mathbf{Asm} construction yields a 2-functor $\mathit{TPCA} \rightarrow \Gamma\nabla\mathit{REG}$ which is locally an equivalence and extends the 2-functor $\mathit{PCA} \rightarrow \Gamma\nabla\mathit{REG}$ described earlier. As a corollary, the categories $\mathbf{Asm}(A)$, $\mathbf{Asm}(B)$ are equivalent if $A \simeq B$ in the 2-category TPCA . (Note that equivalences are possible between TPCAs over different type worlds — indeed, an important theme of [13] was the identification of equivalences between particular untyped PCAs and particular simply typed ones.)

3 Computability structures and realizability

3.1 The 2-category $\mathit{CSTRUCT}$

We are now ready to introduce our primary object of interest: the 2-category of *computability structures* (henceforth *C-structures*) and *simulations* between them. We offer this as a relatively broad framework within which an analogue of the above theory naturally goes through. In essence, the notion of computability structure can be seen as a ‘first order flattening’ of the notion of TPCA, in which data values are sharply distinguished from the computable operations that act on them. Furthermore, our computability structures allow computable operations to be multi-valued or ‘non-deterministic’.

Definition 3.1 *A C-structure C consists of the following:*

- a family $|C|$ of inhabited sets (regarded as ‘datatypes’),
- for each $A, B \in |C|$, a set $C[A, B]$ of relations from A to B (regarded as ‘computable operations’, possibly partial and possibly multi-valued)

such that

- for each $A \in |C|$ there exists $i \in C[A, A]$ with such that $i(a, a)$ for all $a \in A$ (we call such an i a superidentity relation on A);

- if $r \in \mathcal{C}[A, B]$ and $s \in \mathcal{C}[B, C]$, there exists $t \in \mathcal{C}[A, C]$ such that if $r(a, b)$ and $s(b, c)$ then $t(a, c)$ (we call such a relation t a supercomposite of r and s).

A few technical remarks on this definition are in order.

Remarks 3.2 (i) It would arguably be more principled to draw a distinction between *names* of types and their sets of values — that is, to define a C-structure over a set T of type names to be a family of sets $(A_\tau \mid \tau \in T)$ together with certain relations between them, in the spirit of Definition 2.7. This would, for example, allow distinct type names to be assigned the same set of values. However, obviously no essential generality is at stake here, since values may simply be relabelled if necessary, and we prefer to avoid additional level of clutter that such a distinction would entail.

(ii) The fact that we only require ‘superidentity’ and ‘supercomposite’ relations is a feature shared with [6], though not with [4] where the ambient categorical framework imposes the existence of genuine identities and composites. In the case of superidentities, the extra generality is probably a mere curiosity, since we do not know of any natural examples where proper identities are not present. In the case of composition, however, the additional generality is certainly significant: as we shall shortly see, there are natural examples in which supercomposites are present but ordinary relational composites are not. (The decision to use these ‘super’ notions in the definition can also be seen as consonant with the decision to use \succeq rather than \simeq in the definition of PCA.)

(iii) For readers familiar with TPCAs, it might appear surprising that our ‘computable operations’ are essentially *extensional* objects: there would seem to be no provision for distinct ‘operations’ that induce the same relation on data values. This is because ‘computable operations’ generalize the notion of TPCA elements only in their role as the left argument in an application $a \cdot b$ — and for this purpose, only their extension is relevant. We shall see later (Definition 5.22) that when computable operations may themselves be represented by data values, intensional distinctions are indeed catered for.

(iv) Let us note how the above definition relates to the approach taken in [4]. There, the basic notion of ‘model of computation’ is that of a category \mathcal{D} equipped with a functor $F : \mathcal{D} \rightarrow \mathcal{C}$ where \mathcal{C} is a *restriction category* (a ‘category of partial maps’ in a certain abstract sense). No real generality is lost by assuming F is faithful. The leading example of a restriction category is the category of sets and partial functions, and in this case (modulo the difference already noted in (ii) above) such models essentially amount to C-structures in which all computable operations are single-valued. (We shall consider this important class of structures in Section 5.2 below, under the name of *deterministic* C-structures.) If \mathcal{C} is taken to be the category of sets and relations, we obtain in essence the class of all C-structures.

Obviously our notion of a C-structure is very general indeed: for instance, given any category \mathcal{C} with a terminal object 1 , we may form the C-structure whose datatypes are the sets $\text{Hom}(1, X)$ for $X \in \mathcal{C}$, with computable operations induced by morphisms in \mathcal{C} via composition. However, our primary interest is in C-structures that model computational processes of some sort. The following selection gives a broad indication of the kinds of examples we have in mind.

Examples 3.3 (i) Any TPCA A over a type world T gives rise to a C-structure \mathbf{A} : let $|\mathbf{A}|$ be the family of sets A_σ where $\sigma \in T$ (relabelling if necessary as per Remark 3.2(i)); and let $\mathbf{A}[A_\sigma, A_\tau]$ be the set of partial functions $A_\sigma \rightarrow A_\tau$ induced by elements of $A_{\sigma \Rightarrow \tau}$.

(ii) Given a labelled transition system with set of states S , set of labels L , and transition relation $\rightarrow \subseteq S \times L \times S$ (with transitive closure \rightarrow^*), we may define a C-structure \mathbf{C} as follows. Take $|\mathbf{C}| = \{S\}$. For $w \in L^*$ any finite sequence of labels, let r_w be the relation $\{(x, y) \mid x \xrightarrow{w} y\}$ on S , and let $\mathbf{C}[S, S]$ be the set $\{r_w \mid w \in L^*\}$. Clearly this is a C-structure.

(iii) Let \mathcal{L} be some programming language or process calculus comprising the following ingredients:

- An inhabited set T of *types*.
- For each type $\sigma \in T$, a set \mathcal{L}_σ of (closed) *terms* of type σ .
- A small category \mathcal{K} whose set of objects is T , and whose morphisms are regarded as *syntactic contexts*, denoted by $K[-]$. This is equipped with a functor $\Theta : \mathcal{K} \rightarrow \mathbf{Set}$ such that $\Theta(\sigma) = \mathcal{L}_\sigma$ for each $\sigma \in T$; we write $\Theta(K[-])(M)$ as $K[M]$.
- For each $\sigma \in T$, a transitive *many-step reduction* relation $\rightarrow \subseteq \mathcal{L}_\sigma \times \mathcal{L}_\sigma$.
- For each $\sigma \in T$, a subset $\mathcal{V}_\sigma \subseteq \mathcal{L}_\sigma$ of terms designated as *values*.

A context $K[-]$ is called an *evaluation context* if for all $M, M' \in \mathcal{L}_\sigma$, $M \rightarrow M'$ implies $K[M] \rightarrow K[M']$. Clearly evaluation contexts form a subcategory of \mathcal{K} . We write $\mathcal{E}_{\sigma\tau}$ for the set of evaluation contexts from σ to τ .

Given such a language \mathcal{L} , we may define a C-structure \mathbf{L} as follows. Let $|\mathbf{L}|$ be the family of all sets \mathcal{V}_σ for $\sigma \in T$ (again relabelling if necessary as per Remark 3.2(i)). For any $K[-] \in \mathcal{E}_{\sigma\tau}$, let $r_K \subseteq \mathcal{V}_\sigma \times \mathcal{V}_\tau$ be the relation $\{(M, N) \mid K[M] \rightarrow N\}$, and take $\mathbf{L}[\mathcal{V}_\sigma, \mathcal{V}_\tau] = \{r_K \mid K[-] \in \mathcal{E}_{\sigma\tau}\}$. It is easy to check that \mathbf{L} is endowed with identities and supercomposition of relations.

From example (iii) we may now see the prototypical reason why a naturally arising C-structure need not be closed under ordinary relational composition. If $K[M] \rightarrow N$ and $L[N] \rightarrow P$ then $L[K[M]] \rightarrow P$, but the term $L[K[M]]$ might also admit reductions to values that do not proceed via any such $L[N]$. For instance, in a process algebra with parallel composition, if $K[-] = Q|-$ and $L[-] = R|-$, then $L[K[M]] = R|Q|M$ may admit reductions that depend on R interacting directly with M .²

We next introduce the notion of a *simulation* of one C-structure in another; this generalizes the notion of applicative morphism introduced in Section 2.1.

Definition 3.4 *Let \mathbf{C}, \mathbf{D} be C-structures.*

(i) *A simulation $\gamma : \mathbf{C} \rightarrow \mathbf{D}$ consist of:*

- *a function $A \mapsto \gamma A : |\mathbf{C}| \rightarrow |\mathbf{D}|$,*
- *for each $A \in |\mathbf{C}|$, a total relation γ_A from A to γA*

²As a further example of some historical significance, we mention Kleene's class of (S1)–(S9) computable partial functionals of higher type, which are not closed under ordinary composition but admit supercomposites in the present sense.

such that every relation $r \in \mathbf{C}[A, B]$ is tracked by some $r' \in \mathbf{D}[\gamma A, \gamma B]$: that is,

$$r(a, b) \wedge \gamma_A(a, a') \Rightarrow \exists b'. r'(a', b') \wedge \gamma_B(b, b')$$

(ii) If γ, δ are simulations $\mathbf{C} \multimap \mathbf{D}$, we say γ is transformable to δ , and write $\gamma \preceq \delta$, if for each $A \in |\mathbf{C}|$ there exists $t \in \mathbf{D}[\gamma A, \delta A]$ such that

$$\gamma_A(a, a') \Rightarrow \exists a''. t(a', a'') \wedge \delta_A(a, a'')$$

Remarks 3.5 (i) In defining the notion ‘ r' tracks r ’, we are faced with a choice regarding the treatment of non-deterministic (i.e., multi-valued) relations. In the above definition, we have taken the ‘liberal’ approach of requiring only that every possible behaviour of r on an input a can be mirrored by r' on a corresponding element a' ; there may be additional behaviours of r' on a' with no counterpart in r . In Section 5.1, we shall consider a stricter notion of tracking, in which the range of possible behaviours of r' on a' precisely matches that of r on a . Note that these notions coincide if in \mathbf{D} every relation is single-valued.

(ii) Our definition of simulation is closely related to what is termed a *lax total simulation* in [4, Section 2.2]. Indeed, if one takes \mathbf{C} to be the base category of sets and partial functions, then lax total simulations over \mathbf{C} are basically *single-valued* simulations between the corresponding \mathbf{C} -structures, and the *refinement* relation between such simulations coincides with our \preceq . The restriction to single-valued simulations in [4] is less limiting than one might suppose: for instance, we shall see in Section 5.3 that equivalence via single-valued simulations coincides with equivalence via arbitrary ones.

Note that if \mathbf{C} is taken to be the category of sets and relations, the connection is less clear-cut: every lax total simulation is a simulation in our sense, but not conversely.

Examples 3.6 (i) Suppose A, B are TPCAs and $\gamma : A \multimap B$ an applicative morphism in the sense of Section 2.2. Let \mathbf{A}, \mathbf{B} be the \mathbf{C} -structures arising from A, B as in Example 3.3(i). Then γ may clearly be viewed as a simulation $\mathbf{A} \multimap \mathbf{B}$. The question of which simulations $\mathbf{A} \multimap \mathbf{B}$ arise from applicative morphisms in this way will be addressed in Section 5.5.

(ii) Suppose (S, L, \rightarrow) and (S', L, \rightarrow') are labelled transition systems over the same set L of labels, and let \mathbf{C}, \mathbf{C}' be the corresponding \mathbf{C} -structures as in Example 3.3(ii). A simulation $\gamma : \mathbf{C} \multimap \mathbf{C}'$ is then just a certain kind of relation $\gamma \subseteq S \times S'$; we say γ *respects labels* if $r'_{w'}$ tracks r_w implies $w' = w$. A label-respecting simulation is thus precisely ‘half a bisimulation’ in the sense of concurrency theory, and γ is a bisimulation iff both γ and γ^{op} are label-respecting simulations. It would be interesting to know whether the preorder \preceq on simulations corresponds to some natural concept in concurrency theory.

(iii) There are numerous examples of *translations* between programming languages or process calculi that can be viewed as \mathbf{C} -structure simulations via the construction of Example 3.3(iii) — for example, the well-known encodings of λ -calculus into π -calculus [18]. The investigation of such examples deserves a paper to itself, and we refrain from pursuing it any further here.

Proposition 3.7 *\mathbf{C} -structures and simulations (ordered by \preceq) constitute a preorder-enriched category, which we denote by $\mathbf{CSTRUCT}$.*

PROOF: The identity simulation on a \mathbf{C} -structure \mathbf{C} is as expected; note that every relation in \mathbf{C} tracks itself. Composition of simulations is given by ‘typewise’ relational

composition: given $\gamma : C \rightarrow D$ and $\delta : D \rightarrow E$, the action of $\delta \circ \gamma$ on $|C|$ is just the composition of δ and γ , and for each $A \in |C|$, the relation $(\delta \circ \gamma)_A$ is simply the (forwards) relational composition $\gamma_A; \delta_{\gamma A}$. To see that $\delta \circ \gamma$ is indeed a simulation, suppose $r \in C[A, B]$. Take $r' \in D[\gamma A, \gamma B]$ tracking r with respect to γ , and $r'' \in E[\delta \gamma A, \delta \gamma B]$ tracking r' with respect to δ ; then clearly r'' tracks r with respect to $\delta \circ \gamma$. The unit and associativity laws for simulations are now immediate from the corresponding facts for relations.

Next we check that for any C and D , the simulations $C \multimap D$ are preordered by \preceq . For any such simulation γ , the transformation $\gamma \preceq \gamma$ is witnessed at each $A \in C$ by any superidentity relation $i \in D[\gamma A, \gamma A]$. Now suppose $\gamma \preceq \delta \preceq \epsilon$. For any $A \in |C|$, take $t \in D[\gamma A, \delta A]$, $u \in D[\delta A, \epsilon A]$ witnessing $\gamma \preceq \delta$, $\delta \preceq \epsilon$ respectively at A , and let $v \in D[\gamma A, \epsilon A]$ be any supercomposite of t and u . Then v witnesses $\gamma \preceq \epsilon$ at A .

Finally, we show that composition of simulations is monotone in both arguments with respect to \preceq . Firstly, given $\gamma : C \multimap D$ and $\delta \preceq \delta' : D \multimap E$, if t witnesses $\delta \preceq \delta'$ at γA then clearly t also witnesses $\delta \circ \gamma \preceq \delta' \circ \gamma$ at A . Secondly, given $\gamma \preceq \gamma' : C \multimap D$ and $\delta : D \multimap E$, if $t \in D[\gamma A, \gamma' A]$ witnesses $\gamma \preceq \gamma'$ at A then we may take $t' \in E[\delta \gamma A, \delta \gamma' A]$ tracking t with respect to δ ; we claim t' witnesses $\delta \circ \gamma \preceq \delta \circ \gamma'$ at A . For suppose $(\delta \circ \gamma)_A(a, a')$; then there exists a^* with $\gamma_A(a, a^*)$ and $\delta_{\gamma A}(a^*, a')$, so there exists a^{**} with $t(a^*, a^{**})$ and $\gamma'(a, a^{**})$. Hence there exists a'' with $t'(a', a'')$ and $\delta(a^{**}, a'')$, whence $(\delta \circ \gamma')_A(a, a'')$ as required. \square

As we have seen, TPCAs give rise to C-structures and applicative morphisms give rise to simulations between them. Since the preorder \preceq on applicative morphisms clearly agrees with that on simulations, we obtain an inclusion 2-functor $TPCA \rightarrow CSTRUCT$.

3.2 The Asm construction on CSTRUCT

The ‘category of assemblies’ construction can be naturally generalized to C-structures.

Definition 3.8 *Given a C-structure C , we define a category $\mathbf{Asm}(C)$ and a functor $\Gamma_C : \mathbf{Asm}(C) \rightarrow \mathbf{Set}$ as follows:*

- An object X in $\mathbf{Asm}(C)$ is a triple $(|X|, A_X, \Vdash_X)$ where $|X|$ is a set, $A_X \in |C|$, and $\Vdash_X \subseteq A_X \times |X|$ is a relation such that for every $x \in |X|$ there exists some $a \Vdash_X x$.
- A morphism $f : X \rightarrow Y$ in $\mathbf{Asm}(C)$ is a function $f : |X| \rightarrow |Y|$ that is tracked by some $r \in C[A_X, A_Y]$: that is,

$$a \Vdash_X x \Rightarrow \exists b. r(a, b) \wedge b \Vdash_Y f(x)$$

Identities and composition are defined as in \mathbf{Set} .

- Γ_C is the evident forgetful operation $X \mapsto |X|$, $f \mapsto f$.

It is easy to see that $\mathbf{Asm}(C)$ is indeed a category and Γ_C is a faithful functor. We shall always think of $\mathbf{Asm}(C)$ as coming equipped with the functor Γ_C ; in contrast to the situation for PCAs, it will not be the case in general that Γ_C can be recovered from $\mathbf{Asm}(C)$ as the functor $\text{Hom}(1, -)$.

Once again, our notion of ‘tracking’ here involves a choice regarding the treatment of non-determinism. The choice of definition above is consonant with Definition 3.4; when we

consider a stricter notion of simulation in Section 5.1, a more restricted notion of assembly morphism will be appropriate.

We now investigate what structure is present in $(\mathbf{Asm}(\mathcal{C}), \Gamma_{\mathcal{C}})$. For this purpose we introduce the following general notions.

Definition 3.9 *Let \mathcal{C} be a category, Γ a faithful functor $\mathcal{C} \rightarrow \mathbf{Set}$.*

(i) (\mathcal{C}, Γ) has subobjects *if for any object X in \mathcal{C} and any mono $s : S \rightarrow \Gamma(X)$ in \mathbf{Set} , there exists a morphism $\bar{s} : Y \rightarrow X$ in \mathcal{C} (necessarily mono) such that $\Gamma(\bar{s}) = s$, and for any morphism $f : Z \rightarrow X$ such that $\Gamma(f)$ factors through s , there is a unique $g : Z \rightarrow Y$ such that $f = \bar{s} \circ g$.*

(ii) (\mathcal{C}, Γ) has quotients *if for any object X in \mathcal{C} and any epi $q : \Gamma(X) \rightarrow Q$ in \mathbf{Set} , there exists a morphism $\bar{q} : X \rightarrow V$ in \mathcal{C} (necessarily epi) such that $\Gamma(\bar{q}) = q$, and for any morphism $f : X \rightarrow W$ such that $\Gamma(f)$ factors through q , there is a unique $g : V \rightarrow W$ such that $f = g \circ \bar{q}$.*

(iii) (\mathcal{C}, Γ) has copies *if for any $X \in \mathcal{C}$ and $S \in \mathbf{Set}$, there is an object $X \times S$ in \mathcal{C} equipped with morphisms*

$$\pi : X \times S \rightarrow X \quad \theta : \Gamma(X \times S) \rightarrow S$$

such that for any $f : Z \rightarrow X$ and $\phi : \Gamma(Z) \rightarrow S$ there is a unique $g : Z \rightarrow X \times S$ with $f = \pi \circ g$ and $\phi = \theta \circ \Gamma(g)$.

(iv) *We say (\mathcal{C}, Γ) is a quasi-regular category over \mathbf{Set} if it has subobjects, quotients and copies.*

Note that the universal structures given by this definition, when they exist, are unique up to unique isomorphism. In the context of part (i), any morphism \bar{s} with the required properties is called a *lifting* of s to \mathcal{C} ; likewise, in part (ii), we say \bar{q} is a lifting of q to \mathcal{C} . In part (iii), an object $X \times S$ with the required properties is called an *S -fold copy of X* .

The name ‘quasi-regular’ reflects the idea that this is as much of the structure of a regular category as survives in this general setting (where we do not have finite products, for instance). The existence of copies does duty for the functor $\nabla : \mathbf{Set} \rightarrow \mathbf{Asm}(A)$ in the PCA case.

Proposition 3.10 *$(\mathbf{Asm}(\mathcal{C}), \Gamma_{\mathcal{C}})$ is a quasi-regular category over \mathbf{Set} .*

PROOF: To see that $(\mathbf{Asm}(\mathcal{C}), \Gamma_{\mathcal{C}})$ has subobjects, suppose $X \in \mathbf{Asm}(\mathcal{C})$ and $s : S \rightarrow |X|$. Let Y be the assembly defined by $|Y| = S$, $A_Y = A_X$, and $a \Vdash_Y y$ iff $a \Vdash_X s(y)$; and let $\bar{s} = s$ (this is tracked by any superidentity for A_X). The universal property is then easy.

For quotients, suppose $X \in \mathbf{Asm}(\mathcal{C})$ and $q : |X| \rightarrow Q$. Let V be the assembly defined by $|V| = Q$, $A_V = A_X$, and $a \Vdash_V v$ iff there exists $x \in |X|$ with $a \Vdash_X x$ and $q(x) = v$; and let $\bar{q} = q$ (this is tracked by any superidentity for A_X). The universal property is easy.

For copies, suppose $X \in \mathbf{Asm}(\mathcal{C})$ and $S \in \mathbf{Set}$. Define the assembly $X \times S$ by

$$|X \times S| = |X| \times S \quad A_{X \times S} = A_X \quad a \Vdash_{X \times S} (x, s) \text{ iff } a \Vdash_X x.$$

Let $\pi : X \times S \rightarrow X$ be the first projection (this is tracked by any superidentity), and let $\theta : |X \times S| \rightarrow S$ be the second projection. Once again, the universal property is easy. \square

Next, we see how the **Asm** construction may be extended to simulations.

Definition 3.11 Given a simulation $\gamma : \mathbf{C} \multimap \mathbf{D}$, define a functor $\gamma_* = \mathbf{Asm}(\gamma) : \mathbf{Asm}(\mathbf{C}) \rightarrow \mathbf{Asm}(\mathbf{D})$ as follows:

- On objects X , define $\gamma_*(X)$ by $|\gamma_*(X)| = |X|$, $A_{\gamma_*(X)} = \gamma A_X$, and $b \Vdash_{\gamma_*(X)} x$ iff $\exists a \in A_X. \gamma A_X(a, b) \wedge a \Vdash_X x$.
- On morphisms $f : X \rightarrow Y$, define $\gamma_*(f) = f$. Note that if f is tracked by $r \in \mathbf{C}[A_X, A_Y]$, and r is itself tracked (with respect to γ) by $r' \in \mathbf{D}[\gamma A_X, \gamma A_Y]$, then $f : \gamma_*(X) \rightarrow \gamma_*(Y)$ is tracked by r' .

Proposition 3.12 (i) $\mathbf{Asm}(\gamma)$ is indeed a functor $\mathbf{Asm}(\mathbf{C}) \rightarrow \mathbf{Asm}(\mathbf{D})$, and $\Gamma_{\mathbf{D}} \circ \mathbf{Asm}(\gamma) = \Gamma_{\mathbf{C}}$.

(ii) $\mathbf{Asm}(\gamma)$ preserves subobjects, quotients, and copies.

PROOF: All parts are trivial; indeed, the structures described in the proof of Proposition 3.10 are preserved on the nose by $\mathbf{Asm}(\gamma)$. \square

Finally, we show how **Asm** acts on transformations between simulations.

Definition 3.13 Given simulations $\gamma \preceq \delta : \mathbf{C} \multimap \mathbf{D}$, let $\mathbf{Asm}(\gamma \preceq \delta)$ denote the natural transformation $\xi : \mathbf{Asm}(\gamma) \rightarrow \mathbf{Asm}(\delta)$ defined by $\xi_X = \text{id}_{|X|} : \gamma_*(X) \rightarrow \delta_*(X)$. (Note that if $t \in \mathbf{D}[\gamma A_X, \delta A_X]$ witnesses $\gamma \preceq \delta$ at A_X , then t tracks ξ_X .)

Since $\mathbf{Asm}(\gamma \preceq \delta)$ is simply the identity at the level of sets, it is clear that it is indeed a natural transformation.

We may now put all this together.

Definition 3.14 Let $\Gamma\mathcal{QREG}$ be the (large) 2-category defined as follows:

- Objects are quasi-regular categories over **Set**.
- Morphisms $(\mathbf{C}, \Gamma_{\mathbf{C}}) \rightarrow (\mathbf{D}, \Gamma_{\mathbf{D}})$ are functors $F : \mathbf{C} \rightarrow \mathbf{D}$ equipped with a natural isomorphism $\iota : \Gamma_{\mathbf{C}} \cong \Gamma_{\mathbf{D}} \circ F$ such that
 - F preserves subobjects modulo ι : that is, if $\bar{s} : Y \rightarrow X$ is a subobject lifting of $s : S \rightarrow \Gamma_{\mathbf{C}}(X)$, then $F(\bar{s}) : F(Y) \rightarrow F(X)$ is a subobject lifting of $\iota_X \circ s \circ \iota_Y^{-1}$;
 - F preserves quotients modulo ι (the definition is similar);
 - F preserves copies modulo ι : that is, if $(X \times_S S, \pi, \theta)$ is an S -fold copy of X in \mathbf{C} , then $(F(X \times_S S), F(\pi), \theta \circ \iota_{X \times_S S}^{-1})$ is an S -fold copy of $F(X)$ in \mathbf{D} .

We call such a pair (F, ι) a quasi-regular functor.

- 2-cells $(F, \iota) \rightarrow (G, j)$ are natural transformations $\alpha : F \rightarrow G$.

The following proposition addresses a couple of fine details. Again, we use $*$ to denote horizontal composition.

Proposition 3.15 (i) For a quasi-regular functor (F, ι) , the isomorphism ι is uniquely determined by F (so that ι may be dropped from the data for a quasi-regular functor).

(ii) If α is any natural transformation $F \rightarrow G$ where (F, ι) and (G, j) are quasi-regular functors $(\mathcal{C}, \Gamma_{\mathcal{C}}) \rightarrow (\mathcal{D}, \Gamma_{\mathcal{D}})$, then automatically $(\Gamma_{\mathcal{D}} * \alpha) \circ \iota = j$.

PROOF: (i) Suppose $X \in \mathcal{C}$ and $x \in \Gamma_{\mathcal{C}}(X)$; we wish to show that $\iota_X(x)$ is uniquely determined. Let 1 be a singleton set and regard x as a morphism $1 \rightarrow \Gamma_{\mathcal{C}}(X)$; lift this to a morphism $\bar{x} : Y \rightarrow X$ in \mathcal{C} . Since F preserves subobjects modulo ι , we have that $F(\bar{x}) : F(Y) \rightarrow F(X)$ is a subobject lifting of $\iota_X \circ x \circ \iota_Y$. But the latter is a mapping from a singleton set to $\Gamma_{\mathcal{D}}(F(X))$ that picks out $\iota_X(x)$. Hence $\iota_X(x)$ must be the element picked out by $\Gamma_{\mathcal{D}}(F(\bar{x}))$.

(ii) Suppose $X \in \mathcal{C}$ and $x \in \Gamma_{\mathcal{C}}(X)$; we wish to show that $\Gamma_{\mathcal{D}}(\alpha_X)(\iota_X(x)) = j_X(x)$. Again, regard x as a morphism $1 \rightarrow \Gamma_{\mathcal{C}}(X)$ and lift this to a morphism $\bar{x} : Y \rightarrow X$ in \mathcal{C} . As in the proof of (i) we have that $\Gamma_{\mathcal{D}}(F(\bar{x}))$ picks out $\iota_X(x)$ and $\Gamma_{\mathcal{D}}(G(\bar{x}))$ picks out $j_X(x)$. Now applying $\Gamma_{\mathcal{D}}$ to the naturality square for α at \bar{x} , we have that $\Gamma_{\mathcal{D}}(\alpha_X) \circ \Gamma_{\mathcal{D}}(F(\bar{x})) = \Gamma_{\mathcal{D}}(G(\bar{x})) \circ \Gamma_{\mathcal{D}}(\alpha_Y)$; and since $\Gamma_{\mathcal{D}}(\alpha_Y)$ is just a function between singleton sets, this says that $\Gamma_{\mathcal{D}}(\alpha_X)(\iota_X(x)) = j_X(x)$ as required. \square

Note that for a quasi-regular functor F , the isomorphism $\Gamma_{\mathcal{D}} \circ F \cong \Gamma_{\mathcal{C}}$ is uniquely determined by the condition that F must preserve subobjects with respect to it.

Theorem 3.16 The constructions of Definitions 3.8, 3.11 and 3.13 constitute a 2-functor

$$\mathbf{Asm} : \mathcal{CSTRUCT} \rightarrow \Gamma\mathcal{QREG}$$

PROOF: In addition to the properties already established, it is trivial that \mathbf{Asm} respects identity simulations, composition of simulations, identity transformations and vertical and horizontal composition of transformations. \square

Note that the \mathbf{Asm} construction on C-structures essentially agrees with that on TPCAs (as in Section 2.2) modulo the inclusion 2-functor $\mathcal{TPCA} \rightarrow \mathcal{CSTRUCT}$ noted at the end of Section 3.1.

We conclude this section with a brief aside on how our notions of subobject and quotient relate to regular monos and epis in $\mathbf{Asm}(\mathcal{C})$ (we leave the verifications as exercises). On the one hand, all equalizers exist in $\mathbf{Asm}(\mathcal{C})$, and our subobjects are precisely the regular monos. On the other hand, all coequalizers exist in $\mathbf{Asm}(\mathcal{C})$, and every regular epi is a quotient morphism, though the converse does not appear to be true in general. However, if \mathcal{C} is endowed with *cartesian products* (as in Section 5.4 below), then the notions of quotient and regular epi indeed coincide.

Furthermore, for any simulation γ , the functor $\mathbf{Asm}(\gamma)$ preserves all equalizers and coequalizers. In the light of Theorem 3.19 below, this means any quasi-regular functor $\mathbf{Asm}(\mathcal{C}) \rightarrow \mathbf{Asm}(\mathcal{D})$ preserves equalizers and coequalizers.

3.3 The correspondence theorem

We now work towards showing that the 2-functor \mathbf{Asm} is *locally an equivalence*: that is, for any \mathcal{C} and \mathcal{D} , the ordinary functor $\mathbf{Asm} : \mathcal{CSTRUCT}[\mathcal{C}, \mathcal{D}] \rightarrow \Gamma\mathcal{QREG}[\mathbf{Asm}(\mathcal{C}), \mathbf{Asm}(\mathcal{D})]$

is part of an equivalence of preorders. The following proposition shows that essentially every quasi-regular functor $\mathbf{Asm}(\mathbf{C}) \rightarrow \mathbf{Asm}(\mathbf{D})$ arises from some simulation $\mathbf{C} \multimap \mathbf{D}$.

Proposition 3.17 *Let $F : (\mathbf{Asm}(\mathbf{C}), \Gamma_{\mathbf{C}}) \rightarrow (\mathbf{Asm}(\mathbf{D}), \Gamma_{\mathbf{D}})$ be a morphism in $\Gamma\mathcal{QRE}\mathcal{G}$. Then there is a simulation $\gamma : \mathbf{C} \multimap \mathbf{D}$ such that there is a unique natural isomorphism $\mathbf{Asm}(\gamma) \cong F$.*

PROOF: Given F , we define γ as follows. For each $A \in |\mathbf{C}|$, let Z_A be the ‘object of realizers’ in $\mathbf{Asm}(\mathbf{C})$ given by $|Z_A| = A$, $A_{Z_A} = A$, $\Vdash_{Z_A} = \text{id}_A$. Let $W_A = F(Z_A)$, and define $\gamma A = A_{W_A}$ and $\gamma_A(a, b)$ iff $b \Vdash_{W_A} a$.

To see that γ is a simulation, suppose $r \in \mathbf{C}[A, B]$; we wish to show r is tracked by some $r' \in \mathbf{D}[\gamma A, \gamma B]$. Let R be the relation r viewed as a subset of $|Z_A \times B| = A \times B$, and lift this to a subobject $Z_R \rightarrow Z_A \times B$. Likewise, let S be the opposite of r viewed as a subset of $|Z_B \times A| = B \times A$, and lift this to a subobject $Z_S \rightarrow Z_B \times A$. By the universal property of the latter, the canonical bijection $t : R \rightarrow S$ (the twist map) lifts to a morphism $\tilde{t} : Z_R \rightarrow Z_S$ which is tracked by r itself:

$$\begin{array}{ccc} Z_R & \xrightarrow{\tilde{t}} & Z_S \\ \downarrow & & \downarrow \\ Z_A \times B & & Z_B \times A \end{array}$$

We now apply F to this whole diagram, then replace the relevant copies and subobjects by the isomorphic ‘canonical’ ones in $\mathbf{Asm}(\mathbf{D})$ (i.e., those described in the proof of Proposition 3.10):

$$\begin{array}{ccc} Z'_R & \xrightarrow{\hat{t}} & Z'_S \\ \downarrow & & \downarrow \\ F(Z_A) \times B & & F(Z_B) \times A \end{array}$$

Note that the type of realizers for Z'_R is that of $F(Z_A)$, that is γA , and likewise the type of realizers for Z'_S is γB . So let $r' \in \mathbf{D}[\gamma A, \gamma B]$ be any relation tracking \hat{t} . We claim that r' tracks r with respect to γ . Indeed, suppose $r(a, b)$ and $\gamma_A(a, a')$. Then $a' \Vdash_{Z'_R} (a, b)$, since γ_A is by definition extracted from $F(Z_A)$. So there exists $b' \Vdash_{Z'_S} (b, a) = \hat{t}(a, b)$ such that $r'(a', b')$. But now $\gamma_B(b, b')$ by definition of $\Vdash_{Z'_S}$ and γ_B (both are induced by $F(Z_B)$) and we are done.

It remains to show that $\gamma_* = \mathbf{Asm}(\gamma) \cong F$. Consider an arbitrary $X \in \mathbf{Asm}(\mathbf{C})$. We may express X as a quotient of a subobject of $Z_A \times |X|$, where $A = A_X$. Specifically, let R be the subset of $|Z_A \times |X||$ corresponding to \Vdash_X , and let $q : R \twoheadrightarrow |X|$ be the surjection given by second projection. These lift to subobject and quotient maps in $\mathbf{Asm}(\mathbf{C})$, giving

the diagram

$$\begin{array}{ccc} Z_R & \longleftrightarrow & Z_A \times |X| \\ \downarrow & & \\ X & & \end{array}$$

Applying F to this diagram and replacing copies, subobjects and quotients in turn by isomorphic canonical ones, we obtain a diagram

$$\begin{array}{ccc} Z'_R & \longleftrightarrow & F(Z_A) \times |X| \\ \downarrow & & \\ \gamma_*(X) & & \end{array}$$

So we obtain an isomorphism $\gamma_*(X) \cong F(X)$ which agrees on underlying sets with the given isomorphism $\Gamma_C \cong \Gamma_D \circ F$.

To see that this isomorphism is natural in X , it now suffices to note that if $f : X \rightarrow Y$ in $\mathbf{Asm}(\mathbf{C})$ then $\Gamma_D(\gamma_*(f)) = f$, and $\Gamma_D(F(f)) \cong \Gamma_C(f) = f$ via the given isomorphism $\Gamma_D \circ F \cong \Gamma_C$. \square

We now see how the correspondence works at the level of 2-cells:

Proposition 3.18 *Suppose $\gamma, \delta : \mathbf{C} \rightarrow \mathbf{D}$ are simulations, and suppose $\eta : \gamma_* \rightarrow \delta_*$ is a natural transformation. Then $\gamma \preceq \delta$ and $\eta = \mathbf{Asm}(\gamma \preceq \delta)$.*

PROOF: First note that by Proposition 3.15(ii), it is automatic that $\Gamma_D * \eta = \text{id}_{\Gamma_C}$.

For each $A \in |\mathbf{C}|$, let Z_A be the object of realizers as in the previous proof, and consider $\eta_{Z_A} : \gamma_*(Z_A) \rightarrow \delta_*(Z_A)$. Suppose this is tracked by $t \in \mathbf{D}[\gamma A, \delta A]$. Since η_{Z_A} is just the identity on A , we have

$$a' \Vdash_{\gamma_*(Z_A)} a \Rightarrow \exists a'' . t(a', a'') \wedge a'' \Vdash_{\delta_*(Z_A)} a$$

That is, $\gamma(a, a')$ implies $\exists a'' . t(a', a'') \wedge \delta(a, a'')$. In other words, t witnesses $\gamma \preceq \delta$ at A .

Again by Proposition 3.15(ii), it is automatic that $\Gamma_D * \mathbf{Asm}(\gamma \preceq \delta) = \text{id}_{\Gamma_C}$. So $\Gamma_D * \eta = \Gamma_D * \mathbf{Asm}(\gamma \preceq \delta)$, and because Γ_D is faithful, this means $\eta = \mathbf{Asm}(\gamma \preceq \delta)$. \square

We now have everything that is needed for the following, which to some extent validates our definition of *CSTRUCT*:

Theorem 3.19 *The 2-functor $\mathbf{Asm} : \mathbf{CSTRUCT} \rightarrow \Gamma\mathbf{QREG}$ is locally an equivalence. \square*

Corollary 3.20 *The C-structures \mathbf{C} and \mathbf{D} are equivalent in *CSTRUCT* iff there is an equivalence of categories $(F, G) : \mathbf{Asm}(\mathbf{C}) \simeq \mathbf{Asm}(\mathbf{D})$ with $\Gamma_D \circ F \cong \Gamma_C$ and $\Gamma_C \circ G \cong \Gamma_D$.*

PROOF: The forwards implication is immediate from Theorem 3.16 (indeed, this yields and equivalence that commutes with Γ on the nose). For the reverse, it is easy to see that if F and G are as above then F and G are quasi-regular functors, and so constitute an equivalence in $\Gamma\mathbf{QREG}$. It now follows from Theorem 3.19 that $\mathbf{C} \simeq \mathbf{D}$ in *CSTRUCT*. \square

4 Categorical structure in $\mathcal{CSTRUCT}$

We now investigate some of the categorical structure available in the 2-category $\mathcal{CSTRUCT}$. Although our investigation here is far from exhaustive, the picture that emerges is that $\mathcal{CSTRUCT}$ is considerably richer in structure than either \mathcal{PCA} or \mathcal{TPCA} .

If X and Y are sets of sets, we write $X \boxtimes Y$ for the set $\{A \times B \mid A \in X, B \in Y\}$.

Proposition 4.1 $\mathcal{CSTRUCT}$ has finite products, enriched with respect to \preceq .

PROOF: The terminal object $\mathbf{1}$ is the unique C-structure with $|\mathbf{1}| = \{\{*\}\}$. If \mathbf{C}, \mathbf{D} are C-structures, define $\mathbf{C} \times \mathbf{D}$ by

$$\begin{aligned} |\mathbf{C} \times \mathbf{D}| &= |\mathbf{C}| \boxtimes |\mathbf{D}| \\ (\mathbf{C} \times \mathbf{D})[A \times B, C \times D] &= \mathbf{C}[A, C] \boxtimes \mathbf{D}[B, D] \end{aligned}$$

Clearly $\mathbf{C} \times \mathbf{D}$ is a C-structure, and there are evident projection simulations $\pi_{\mathbf{C}} : \mathbf{C} \times \mathbf{D} \rightarrow \mathbf{C}$ and $\pi_{\mathbf{D}} : \mathbf{C} \times \mathbf{D} \rightarrow \mathbf{D}$. Given simulations $\gamma : \mathbf{E} \rightarrow \mathbf{C}$ and $\delta : \mathbf{E} \rightarrow \mathbf{D}$, we have a simulation $\langle \gamma, \delta \rangle : \mathbf{E} \rightarrow \mathbf{C} \times \mathbf{D}$ given by

$$\begin{aligned} \langle \gamma, \delta \rangle A &= \gamma A \times \delta A \\ \langle \gamma, \delta \rangle_A(e, (c, d)) &\Leftrightarrow \gamma_A(e, c) \wedge \delta_A(e, d) \end{aligned}$$

Clearly this is the unique simulation such that $\pi_{\mathbf{C}} \circ \langle \gamma, \delta \rangle = \gamma$ and $\pi_{\mathbf{D}} \circ \langle \gamma, \delta \rangle = \delta$, and $\langle \gamma, \delta \rangle \preceq \langle \gamma', \delta' \rangle$ iff $\gamma \preceq \gamma'$ and $\delta \preceq \delta'$. \square

Proposition 4.2 $\mathcal{CSTRUCT}$ has finite sums, enriched with respect to \preceq .

PROOF: The initial object $\mathbf{0}$ is the empty C-structure. As a temporary notation, for any set A we write A_i for $\{i\} \times A$. If \mathbf{C}, \mathbf{D} are C-structures, define $\mathbf{C} + \mathbf{D}$ by

$$\begin{aligned} |\mathbf{C} + \mathbf{D}| &= \{A_0 \mid A \in |\mathbf{C}|\} \cup \{B_1 \mid B \in |\mathbf{D}|\} \\ (\mathbf{C} + \mathbf{D})[A_0, B_0] &\cong \mathbf{C}[A, B] \\ (\mathbf{C} + \mathbf{D})[A_1, B_1] &\cong \mathbf{D}[A, B] \\ (\mathbf{C} + \mathbf{D})[A_i, B_j] &= \emptyset \text{ if } i \neq j \end{aligned}$$

Clearly $\mathbf{C} + \mathbf{D}$ is a C-structure, and there are evident inclusion simulations $\iota_{\mathbf{C}} : \mathbf{C} \rightarrow \mathbf{C} + \mathbf{D}$ and $\iota_{\mathbf{D}} : \mathbf{D} \rightarrow \mathbf{C} + \mathbf{D}$. Given simulations $\gamma : \mathbf{C} \rightarrow \mathbf{E}$ and $\delta : \mathbf{D} \rightarrow \mathbf{E}$, there is evidently a unique simulation $[\gamma, \delta] : \mathbf{C} + \mathbf{D} \rightarrow \mathbf{E}$ such that $[\gamma, \delta] \circ \iota_{\mathbf{C}} = \gamma$ and $[\gamma, \delta] \circ \iota_{\mathbf{D}} = \delta$, and clearly $[\gamma, \delta] \preceq [\gamma', \delta']$ iff $\gamma \preceq \gamma'$ and $\delta \preceq \delta'$. \square

Although both these results are trivial, note that Proposition 4.2 already gives us something not available in \mathcal{PCA} or \mathcal{TPCA} . If A, B were TPCAs over some T , for instance, the evident disjoint union $A + B$ would not (canonically) be a TPCA, since it would lack a type of operations mapping A_σ to B_τ (where $\sigma, \tau \in T$). Most probably, \mathcal{PCA} and \mathcal{TPCA} do not have categorical binary sums at all.

More interestingly, it turns out that $\mathcal{CSTRUCT}$ is very nearly cartesian closed: indeed, for any C-structures \mathbf{C}, \mathbf{D} , an ‘exponential’ C-structure $\mathbf{D}^{\mathbf{C}}$ exists in a slightly weak sense which is nonetheless sufficient to characterize $\mathbf{D}^{\mathbf{C}}$ uniquely up to equivalence in $\mathcal{CSTRUCT}$. Let us begin by making the characteristic property of $\mathbf{D}^{\mathbf{C}}$ precise.

Definition 4.3 A simulation $\gamma : C \multimap D$ is single-valued if for any $A \in |C|$ and $a \in A$, there is a unique $a' \in \gamma A$ with $\gamma_A(a, a')$.

Definition 4.4 Suppose C, D are C -structures, and we are given a C -structure F along with a simulation $eval : F \times C \multimap D$. We call $(F, eval)$ a near-exponential for (C, D) if for any C -structure E and simulation $\alpha : E \times C \multimap D$, there is a single-valued simulation $\bar{\alpha} : E \multimap F$ such that $eval \circ (\bar{\alpha} \times id_C) = \alpha$, and moreover $\bar{\alpha}$ is, up to \preceq, \succeq , the unique single-valued simulation with this property.

Because single-valued simulations compose and identity simulations are single-valued, it is easy to see that a near-exponential for (C, D) is uniquely determined up to equivalence in $CSTRUCT$.

The rest of this section will be devoted to the proof of the following:

Theorem 4.5 $CSTRUCT$ possesses a near-exponential $(D^C, eval)$ for every (C, D) .

The conceptual significance of this result is far from clear to us: if C and D represent ‘models of computation’, what on earth does it mean to consider D^C as a ‘model of computation’? At present, the existence of near-exponential stands as something of a curiosity, albeit a rather striking one which offers evidence for the mathematical richness of our framework.

From now on let C and D be fixed C -structures. We start with the definition of the C -structure D^C .

Definition 4.6 (i) Given any function $U : |C| \rightarrow |D|$, we write $[C \multimap_U D]$ for the set of simulations $\gamma : C \multimap D$ such that $\gamma A = U(A)$ for all $A \in |C|$.

(ii) A family $F \subseteq [C \multimap_U D]$ of simulations is uniformly tracked if for all $A, B \in |C|$ and $r \in C[A, B]$, there exists $r' \in D[U(A), U(B)]$ such that r' tracks r with respect to every $\gamma \in F$: that is,

$$\forall \gamma \in F. r(a, b) \wedge \gamma_A(a, a') \Rightarrow \exists b'. r'(a', b') \wedge \gamma_B(b, b')$$

(Note that if F is uniformly tracked and inhabited then we can recover U from F ; we write it as U_F and call it the underlying type operation of F .)

(iii) If $F \subseteq [C \multimap_U D]$ and $G \subseteq [C \multimap_V D]$ are each uniformly tracked and inhabited, we say $R \subseteq F \times G$ is a uniform transformation if for all $A \in |C|$ there exists $t \in D[U(A), V(A)]$ such that for every $(\gamma, \delta) \in R$ we have that t witnesses $\gamma \preceq \delta$ at A : that is,

$$\forall (\gamma, \delta) \in R. \gamma_A(a, b) \Rightarrow \exists c. t(b, c) \wedge \delta_A(a, c)$$

(iv) We now define a C -structure D^C as follows:

- $|D^C|$ is the set of all uniformly tracked and inhabited families F of simulations $C \multimap D$.
- For any $F, G \in |D^C|$, $D^C[F, G]$ is the set of uniform transformations $R \subseteq F \times G$.

Proposition 4.7 D^C is indeed a C -structure.

PROOF: To see that D^C has (super)identities, given any $F \in |D^C|$, take $I \subseteq F \times F$ to be the identity relation. Clearly I is a uniform transformation: for any $A \in |C|$, let i be any superidentity on $U_F(A)$; then for any $\gamma \in F$, i witnesses $\gamma \preceq \gamma$ at A . So $I \in D^C[F, F]$.

To see that D^C has (super)composites, suppose $R \in D^C[F, G]$ and $S \in D^C[G, H]$ and let T be the relational composition $R; S$. We claim T is a uniform transformation. Given $A \in |C|$, take $t \in D[U_F(A), U_G(A)]$ uniformly witnessing $\gamma \preceq \delta$ at A for all $(\gamma, \delta) \in R$, and take $u \in D[U_G(A), U_H(A)]$ uniformly witnessing $\delta \preceq \epsilon$ at A for all $(\delta, \epsilon) \in S$. Let $v \in D[U_F(A), U_H(A)]$ be any supercomposite of t and u ; clearly v uniformly witnesses $\gamma \preceq \epsilon$ at A for all $(\gamma, \epsilon) \in T$. \square

Next, we must equip D^C with a suitable evaluation morphism.

Definition 4.8 *Let $eval : D^C \times C \rightarrow D$ be the simulation given as follows.*

- The action on types is given by $eval(F \times A) = U_F(A)$.
- For each F and A , we define $eval_{F \times A} \subseteq (F \times A) \times U_F(A)$ by

$$eval_{F \times A}((\gamma, a), b) \Leftrightarrow \gamma_A(a, b)$$

(This is a total relation since each γ_A is total.)

Proposition 4.9 *$eval : D^C \times C \rightarrow D$ is indeed a simulation.*

PROOF: We wish to show that every relation $R \times r$ in $(D^C \times C)[F \times A, G \times B] = D^C[F, G] \boxtimes C[A, B]$ is tracked by some $u \in D[U_F(A), U_G(B)]$ with respect to $eval$. Let $s \in D[U_F(A), U_F(B)]$ track r uniformly with respect to all $\gamma \in F$, and let $t \in D[U_F(B), U_G(B)]$ witness $\gamma \preceq \delta$ at B uniformly for all $(\gamma, \delta) \in R$. Now let u be a supercomposite of s and t .

To see that u tracks $R \times r$ with respect to $eval$, suppose that $((\gamma, a), (\delta, b)) \in (R \times r)$ and $eval((\gamma, a), a')$ — that is to say, $(\gamma, \delta) \in R$, $(a, b) \in r$ and $\gamma_A(a, a')$. We require $b' \in \delta B$ such that $u(a', b')$ and $eval((\delta, b), b')$ — that is, $\delta_B(b, b')$. Since s tracks r with respect to γ , we may find b'' such that $s(a, b'')$ and $\gamma_B(b, b'')$. And since t witnesses $\gamma \preceq \delta$ at B , we may find $t(b'', b')$ and $\delta_B(b, b')$. This gives $u(a', b')$ as required. \square

It remains to verify that $(D^C, eval)$ is a near-exponential for (C, D) . From now on, let E be some fixed C -structure and $\alpha : E \times C \rightarrow D$ some fixed simulation.

For any $E \in |E|$ and any $e \in E$, let us write α_e for the simulation $C \rightarrow D$ defined by

$$\alpha_e A = \alpha(E \times A) \quad \alpha_{eA}(c, d) \Leftrightarrow \alpha_{E \times A}((e, c), d)$$

To see that α_e is a simulation, suppose $r \in C[A, B]$. Let i be a superidentity in $E[E, E]$; then any $t \in D[\alpha(E \times A), \alpha(E \times B)]$ that tracks $i \times r$ with respect to α also tracks r with respect to α_e . Since such a t may be chosen independently of e , the family $\{\alpha_e \mid e \in E\}$ is uniformly tracked. This justifies the following definition:

Definition 4.10 *Let $\bar{\alpha} : E \rightarrow D^C$ be the simulation defined as follows (we call $\bar{\alpha}$ the transpose of α).*

- For each $E \in |E|$, let $\bar{\alpha}E = \{\alpha_e \mid e \in E\} \in |D^C|$.
- For $E \in |E|$ and $e \in E$, take $\bar{\alpha}_E(e, \gamma)$ iff $\gamma = \alpha_e$.

Proposition 4.11 $\bar{\alpha}$ is a single-valued simulation, and $eval \circ (\bar{\alpha} \times id_C) = \alpha$ on the nose.

PROOF: That $\bar{\alpha}$ is single-valued is immediate from the definition. To see that $\bar{\alpha}$ is a simulation, suppose $r \in \mathbf{E}[E, F]$, and let $R = \{(\alpha_e, \alpha_f) \mid (e, f) \in r\}$. We first show that R is a uniform transformation. Given any $A \in |\mathbf{C}|$, let $i \in \mathbf{C}[A, A]$ be a superidentity, and take $r' \in \mathbf{D}[\alpha(E \times A), \alpha(F \times A)]$ tracking $r \times i$ with respect to α . We claim that for any $(e, f) \in r$, r' tracks $\alpha_e \preceq \alpha_f$ at A . For suppose $\alpha_e(c, d)$; then $\alpha((e, c), d)$, so there exists g with $r(d, g)$ and $\alpha((f, c), g)$, whence $\alpha_f(c, g)$. Thus $R \in \mathbf{D}^C[\bar{\alpha}E, \bar{\alpha}F]$.

Next, we claim that R tracks r with respect to $\bar{\alpha}$. For suppose $r(e, f)$ and $\bar{\alpha}(e, \gamma)$; then $\gamma = \alpha_e$, so taking $\delta = \alpha_f$ we have $\bar{\alpha}(f, \delta)$ and $R(\gamma, \delta)$.

To see that $eval \circ (\bar{\alpha} \times id_C) = \alpha$, note that if $E \in |\mathbf{E}|$, $C \in |\mathbf{C}|$ and $D = \alpha(E \times C)$ then

$$\begin{aligned} (eval \circ (\bar{\alpha} \times id_C))_{E \times C}((e, c), d) &\Leftrightarrow eval_{\bar{\alpha}E \times C}((\alpha_e, c), d) \\ &\Leftrightarrow \alpha_{eC}(c, d) \\ &\Leftrightarrow \alpha_{E \times C}((e, c), d) \quad \square \end{aligned}$$

This already establishes that $\mathcal{CSTRUCT}$ is at least *weakly cartesian closed*. To conclude our proof that $\mathcal{CSTRUCT}$ has near-exponentials, we need the following:

Proposition 4.12 Suppose $\bar{\alpha}' : \mathbf{E} \rightarrow \mathbf{D}^C$ is any single-valued simulation such that $eval \circ (\bar{\alpha}' \times id_C) = \alpha$. Then $\bar{\alpha} \preceq \bar{\alpha}' \preceq \bar{\alpha}$.

PROOF: Suppose $\bar{\alpha}'$ is as above. For any $E \in |\mathbf{E}|$ and $e \in E$, write α'_e for the unique simulation $\mathbf{C} \rightarrow \mathbf{D}$ such that $\bar{\alpha}'_E(e, \alpha'_e)$. Then as in the proof of Proposition 4.11, we have $\alpha'_{eC}(c, d)$ iff $\alpha_{eC}(c, d)$. So $\alpha'_e = \alpha_e$, though $\bar{\alpha}E$ and $\bar{\alpha}'E$ may be different. However, we have shown that $\bar{\alpha}'(e, \alpha_e)$ for each e , so we know that $\bar{\alpha}E = \{\alpha_e \mid e \in E\} \subseteq \bar{\alpha}'E$. So we may consider $\bar{\alpha}E \times \bar{\alpha}E$ as a subset of $\bar{\alpha}E \times \bar{\alpha}'E$, or of $\bar{\alpha}'E \times \bar{\alpha}E$. In either case, $\bar{\alpha}E \times \bar{\alpha}E$ is a uniform transformation, being witnessed by superidentities everywhere, and so is an element of both $\mathbf{D}^C[\bar{\alpha}E, \bar{\alpha}'E]$ and $\mathbf{D}^C[\bar{\alpha}'E, \bar{\alpha}E]$. The first of these witnesses $\bar{\alpha} \preceq \bar{\alpha}'$ at E ; the second $\bar{\alpha}' \preceq \bar{\alpha}$ at E . Since $E \in |\mathbf{E}|$ was arbitrary, we have $\bar{\alpha} \preceq \bar{\alpha}' \preceq \bar{\alpha}$. \square

This completes the proof of Theorem 4.5. However, it is worth noting that a little more may be said about the morphism $\bar{\alpha}$:

Proposition 4.13 Suppose $\bar{\alpha}' : \mathbf{E} \rightarrow \mathbf{D}^C$ is any simulation (not necessarily single-valued) such that $eval \circ (\bar{\alpha}' \times id_C) = \alpha$. Then $\bar{\alpha}' \preceq \bar{\alpha}$.

PROOF: Suppose $\bar{\alpha}'$ is as above. Given $E \in |\mathbf{E}|$ and $e \in E$, let ϵ be any simulation $\mathbf{C} \rightarrow \mathbf{D}$ such that $\bar{\alpha}'_E(e, \epsilon)$. Then we have the sequence of implications

$$\begin{aligned} \epsilon_C(c, d) &\Rightarrow eval_{\bar{\alpha}'E \times C}((\epsilon, c), d) \\ &\Rightarrow (eval \circ (\bar{\alpha}' \times id_C))_{E \times C}((e, c), d) \\ &\Rightarrow \alpha_{E \times C}((e, c), d) \\ &\Rightarrow \alpha_{eC}(c, d) \end{aligned}$$

So at any $C \in |\mathbf{C}|$, $\epsilon \preceq \alpha_e$ is witnessed at C by a superidentity on $\alpha(E \times C)$, independently of e . So let $R \subseteq \bar{\alpha}'E \times \bar{\alpha}E$ be the relation $\{(\epsilon, \alpha_e) \mid e \in E, (e, \epsilon) \in \bar{\alpha}'E\}$; then R is a uniform transformation and witnesses $\bar{\alpha}' \preceq \bar{\alpha}$ at E . Since E is arbitrary, we have $\bar{\alpha}' \preceq \bar{\alpha}$. \square

We remark in passing that even for relatively simple \mathbf{C} and \mathbf{D} , the structure of $\mathbf{D}^{\mathbf{C}}$ may be wildly intractable. For instance, if K_1 denotes Kleene's first model (the PCA of natural numbers with Kleene application) then $K_1^{K_1}$ would appear to be at least as complicated as the lattice of Turing degrees, probably much worse. A possibility for a radically cut-down version of our near-exponentials will be briefly mentioned in Section 6.2.

5 Some subcategories of *CSTRUCT*

There are many natural subcategories of *CSTRUCT* that one might consider. In [12, Chapter 2] we showed how certain specific properties of PCAs and of applicative morphisms were reflected in categorical properties of the corresponding categories and functors; here we show how a similar though somewhat richer story may be told for C-structures and simulations.

5.1 Tightness

We start by sketching the alternative approach to non-determinism that we alluded to earlier. The basic idea here is that if a relation r' is meant to 'simulate' some relation r , it is reasonable to require that r' admits only as much non-determinism as r does; a simulation with this property is called *tight*. As we shall see, a satisfactory analogue of our entire theory may be obtained by consistently 'tightening' all our definitions, and whilst this involves a little more technical detail, in some respects this alternative version enjoys more pleasant properties than the original. (Our use of the term 'tight' is similar in spirit to, though technically different from, its use in [4].)

Definition 5.1 (i) A C-structure \mathbf{C} is called *tight* if

- for any $A \in |\mathbf{C}|$, the identity relation id_A is present in $\mathbf{C}[A, A]$;
- for any $r \in \mathbf{C}[A, B]$ and $s \in \mathbf{C}[B, C]$, there exists $t \in \mathbf{C}[A, C]$ such that

$$a \in \text{dom}(r; s) \Rightarrow (t(a, c) \Leftrightarrow \exists b. r(a, b) \wedge s(b, c))$$

(ii) A simulation $\gamma : \mathbf{C} \multimap \mathbf{D}$ is *tight* if every $r \in \mathbf{C}[A, B]$ is tightly tracked by some $r' \in \mathbf{D}[\gamma A, \gamma B]$: that is, r' tracks r in the usual sense, and

$$r(a, b) \wedge \gamma(a, a') \wedge r'(a', b') \Rightarrow \gamma(b, b')$$

(iii) Given $\gamma, \delta : \mathbf{C} \multimap \mathbf{D}$, we say γ is *tightly transformable* to δ , and write $\gamma \preceq_t \delta$, if for each $A \in |\mathbf{C}|$ there exists $t \in \mathbf{D}[\gamma A, \delta A]$ such that

$$\gamma_A(a, a') \Rightarrow (\exists a''. t(a', a'')) \wedge (\forall a''. t(a', a'') \Rightarrow \delta_A(a, a''))$$

A tight C-structure still need not be closed under relational composition, since in part (i) of the above definition the domain of t may be larger than that of $r; s$. Whilst in principle one could consider tight simulations between non-tight C-structures, there would appear to be little merit in doing so. It is routine to check that tight C-structures, tight simulations

and tight transformations between them constitute a preorder-enriched category, which we denote by $CSTRUCT_t$. We say C, D are *tightly equivalent*, and write $C \simeq_t D$, if they are equivalent in $CSTRUCT_t$.

We may also apply the concept of tightness to morphisms of assemblies.

Definition 5.2 *A morphism $f : X \rightarrow Y$ in $\mathbf{Asm}(C)$ is called tight if it is tightly tracked by some $r \in C[A_X, A_Y]$: that is, r tracks f in the usual sense, and*

$$a \Vdash_X x \wedge r(a, b) \Rightarrow b \Vdash_Y f(x)$$

If C is tight, it is easy to check that assemblies and tight morphisms constitute a wide subcategory of $\mathbf{Asm}(C)$, which we denote by $\mathbf{Asm}_t(C)$. The relationship with tight simulations is given by the following:

Proposition 5.3 (i) *Suppose C, D are tight C-structures. A simulation $\gamma : C \multimap D$ is tight iff $\gamma_* = \mathbf{Asm}(\gamma)$ maps tight morphisms to tight morphisms.*

(ii) *Suppose $\gamma, \delta : C \multimap D$ are tight simulations between tight C-structures, and $\gamma \preceq \delta$. Then $\gamma \preceq_t \delta$ iff all components of $\mathbf{Asm}(\gamma \preceq \delta)$ are tight morphisms.*

PROOF: (i) Suppose γ is tight. If r tightly tracks $f : X \rightarrow Y$ in $\mathbf{Asm}(C)$, and r' tightly tracks r with respect to γ , it is easy to check that r' tightly tracks $\gamma_*(f)$ in $\mathbf{Asm}(D)$. Conversely, suppose γ_* preserves tight morphisms, and suppose $r \in C[A, B]$. Define assemblies X, Y as follows:

$$|X| = |Y| = \text{dom } r, \quad A_X = A, \quad A_Y = B, \quad a' \Vdash_X a \Leftrightarrow a' = a, \quad b \Vdash_Y a \Leftrightarrow r(a, b)$$

Let $f : X \rightarrow Y$ be the identity function on $\text{dom } r$; clearly this is tightly tracked by r itself. So $\gamma_*(f)$ is tightly tracked by some r' , and it is easy to see that this r' tightly tracks r with respect to γ .

(ii) Let $\xi = \mathbf{Asm}(\gamma \preceq \delta)$. First, suppose $\gamma \preceq_t \delta$, and consider the morphism ξ_X for an arbitrary assembly $X \in \mathbf{Asm}(C)$. Take $t \in D[\gamma A_X, \delta A_X]$ that tightly witnesses $\gamma \preceq \delta$ at A_X ; then t tightly tracks ξ_X . Conversely, suppose every ξ_X is tight, and consider an arbitrary $A \in |C|$. Let Z_A be the object of realizers given by $|Z_A| = A, A_{Z_A} = A, \Vdash_{Z_A} = \text{id}_A$, and suppose r tightly tracks ξ_{Z_A} ; then r tightly witnesses $\gamma \preceq \delta$ at A . \square

In fact, by considering $\mathbf{Asm}_t(C)$ as a category in its own right, we may obtain a ‘tight’ analogue of our theory which makes no reference to $\mathbf{Asm}(C)$. If C is tight, then clearly all the subobject, quotient and copy projection morphisms appearing in the proof of Proposition 3.10 are tight, and it is easy to see that the relevant universal properties still hold within $\mathbf{Asm}_t(C)$. So $(\mathbf{Asm}_t(C), \Gamma_C)$ is a quasi-regular category. Moreover, the above proposition shows that if γ is a tight simulation between tight C-structures, then $\mathbf{Asm}(\gamma)$ restricts to a functor $\mathbf{Asm}_t(\gamma) : \mathbf{Asm}_t(C) \rightarrow \mathbf{Asm}_t(D)$, and that if $\gamma \preceq_t \delta$ then $\mathbf{Asm}(\gamma \preceq \delta)$ constitutes a natural transformation between such functors. We thus have a 2-functor

$$\mathbf{Asm}_t : CSTRUCT_t \rightarrow \Gamma QREG$$

Furthermore, the analogue of Theorem 3.19 holds:

Theorem 5.4 *The 2-functor \mathbf{Asm}_t is locally an equivalence. Hence $C \simeq_t D$ iff $\mathbf{Asm}_t(C), \mathbf{Asm}_t(D)$ are equivalent in $\Gamma QREG$.*

PROOF: The proof of Theorem 3.19 goes through in the tight setting with minor adjustments. \square

The categorical structure investigated in Section 4 also carries over to $CSTRUCT_t$. Firstly, $CSTRUCT_t$ clearly inherits the finite products and sums described in Propositions 4.1 and 4.2, so that the inclusion $CSTRUCT_t \rightarrow CSTRUCT$ preserves this structure. Secondly, $CSTRUCT_t$ possesses near-exponentials analogous to, but different from, those of $CSTRUCT$, so that these are not preserved by the inclusion. Specifically, to construct D^C within $CSTRUCT_t$, one should take $|D^C|$ to be the set of all inhabited and uniformly *tightly* tracked families of simulations $C \multimap D$, and $D^C[F, G]$ to be the set of uniform *tight* transformations $R \subseteq F \times G$. The proof of Theorem 4.5 then goes through *mutatis mutandis*.

A few minor gaps in this story should be noted. There does not seem to be a simple categorical characterization of the tight assembly morphisms among those of $\mathbf{Asm}(C)$, nor of those functors that preserve tight morphisms. Likewise, the property that C is tight does not seem to be mirrored by any particular categorical property of $\mathbf{Asm}(C)$; indeed, we conjecture that a tight C -structure may be equivalent in $CSTRUCT$ to a non-tight one, so that tightness is not a genuinely invariant property of the notion of computability represented by C from the perspective of $CSTRUCT$.

5.2 Deterministic C-structures

A class of C -structures it is very natural to consider consists of those that model deterministic flavours of computation. We here show that this is a mathematically well-behaved class from the point of view of our theory.

Definition 5.5 *A C-structure C is deterministic if for any $r \in C[A, B]$ we have*

$$r(a, b) \wedge r(a, b') \Rightarrow b = b'$$

Thus, deterministic C -structures are those in which the ‘relations’ are partial functions. We write $r \cdot a$ for the unique b such that $r(a, b)$ if one exists. For such structures, the tight/non-tight distinction evaporates completely. For example, if D is deterministic then not only is D tight, but every simulation $C \multimap D$ is tight, as is every transformation $\gamma \preceq \delta$ where $\gamma, \delta : C \multimap D$. Furthermore:

Proposition 5.6 *A C-structure D is deterministic iff $\mathbf{Asm}(D) = \mathbf{Asm}_t(D)$.*

PROOF: If D is deterministic and r tracks $f : X \rightarrow Y$ in $\mathbf{Asm}(D)$, then for any $a \Vdash_X x$ there is exactly one b with $r(a, b)$, so r tracks f tightly and $f \in \mathbf{Asm}_t(D)$. Conversely, if D is not deterministic, take $r \in D[A, B]$ and $a \in A, b, b' \in B$ with $r(a, b), r(a, b')$ and $b \neq b'$, and consider the assemblies X, Y defined by

$$|X| = |Y| = \{b, b'\}, \quad A_X = A, \quad A_Y = B, \quad a' \Vdash_X x \Leftrightarrow a' = a, \quad b'' \Vdash_Y y \Leftrightarrow b'' = y$$

Then the identity map $X \rightarrow Y$ is tracked by r , but cannot be tightly tracked by any $r' \in D[A, B]$. \square

Deterministic C-structures thus form a full sub-2-category of both $\mathcal{CSTRUCT}$ and $\mathcal{CSTRUCT}_t$, which we denote by $\mathcal{CSTRUCT}_d$. Moreover, the class of deterministic C-structures is closed under equivalences in $\mathcal{CSTRUCT}_t$:

Proposition 5.7 *If D is deterministic and $C \simeq D$ in $\mathcal{CSTRUCT}_t$, then C is deterministic.*

PROOF: Clearly an equivalence in $\mathcal{CSTRUCT}_t$ is also an equivalence in $\mathcal{CSTRUCT}$, so by the 2-functoriality of \mathbf{Asm}_t and \mathbf{Asm} , if $C \simeq D$ in $\mathcal{CSTRUCT}_t$ then both $\mathbf{Asm}_t(C) \simeq \mathbf{Asm}_t(D)$ and $\mathbf{Asm}(C) \simeq \mathbf{Asm}(D)$. Moreover, the equivalences are easily seen to commute with the inclusions $\mathbf{Asm}_t(-) \rightarrow \mathbf{Asm}(-)$. But if D is deterministic then $\mathbf{Asm}_t(D)$ is the whole of $\mathbf{Asm}(D)$, whence $\mathbf{Asm}_t(C) = \mathbf{Asm}(C)$ and C is deterministic by Proposition 5.6. \square

This says in effect that ‘determinism’ is indeed an invariant property of a computability notion with respect to tight equivalences. We do not know whether this is the case for equivalences in $\mathcal{CSTRUCT}$. Also, it would be interesting to find some simple categorical property of $\mathbf{Asm}_t(D)$ that holds iff D is deterministic.

Deterministic C-structures are generally pleasant to work with since they allow many of the basic definitions to be significantly simplified. For instance, treating relations as partial functions, the notion ‘ r' tracks r with respect to γ ’ may be expressed as

$$\gamma_A(a, a') \wedge r(a) \downarrow \Rightarrow \gamma_B(r \cdot a, r' \cdot a')$$

and ‘ t witnesses $\gamma \preceq \delta$ at A ’ may be expressed as

$$\gamma_A(a, a') \Rightarrow \gamma_A(a, t \cdot a')$$

The definition of morphism in $\mathbf{Asm}(C)$ may similarly be simplified.

Clearly the class of deterministic C-structures is closed under finite products and sums. Unfortunately they are not closed under the near-exponentials of Definition 4.6 (or their tight counterparts); however, it turns out that $\mathcal{CSTRUCT}_d$ is endowed with its own ‘near-exponentials’ in a somewhat weaker sense. Let us here say a simulation $E \times C \multimap D$ is *left-injective* if for any $E \in |\mathbf{E}|$, the simulations α_e for $e \in E$ are all distinct.

Theorem 5.8 *If C, D are deterministic C-structures, there exist a deterministic C-structure F and a left-injective simulation $\text{eval} : F \times C \multimap D$ with the following property: for any left-injective $\alpha : E \times C \multimap D$ there is a single-valued simulation $\bar{\alpha} : E \multimap F$ such that $\text{eval} \circ (\bar{\alpha} \times \text{id}_C) = \alpha$, and moreover $\bar{\alpha}$ is unique up to $\preceq \succeq$ among simulations with this property.*

PROOF SKETCH: Take $|F|$ to be the set of all inhabited, uniformly tracked families of simulations as before, and for $F, G \in |F|$, define $F[F, G]$ to be the set of *single-valued* uniform transformations $R \subseteq F \times G$. The rest of the argument then proceeds essentially as before; the significance of left-injectivity is that in the proof of Proposition 4.11, we need to know that if $r \in F[E, F]$ is single-valued then so is $R = \{(\alpha_e, \alpha_f) \mid (e, f) \in r\}$. \square

Note that even this weaker universal property is still sufficient to characterize the C-structure F up to equivalence, so that we may speak of the near-exponentiation of deterministic C-structures as a well-defined operation on deterministic notions of computability.

5.3 Discrete and univalent morphisms

Among the properties of simulations considered in [12] in the PCA setting were two properties known as *discreteness* and *projectivity*. We here consider the analogues of both these properties, changing the name of the latter to *univalence*, for reasons to be explained. Each of these properties of simulations is naturally correlated with a property of assemblies.

Definition 5.9 (i) A simulation $\gamma : \mathbf{C} \multimap \mathbf{D}$ is called *discrete* if $\gamma_A(a, b)$ and $\gamma_A(a', b)$ imply $a = a'$.

(ii) An assembly X over \mathbf{C} is *discrete* if $a \Vdash_X x$ and $a \Vdash_X x'$ imply $x = x'$. (Such assemblies are also known as *modest sets*.)

The following is straightforward:

Proposition 5.10 A simulation γ is discrete iff $\mathbf{Asm}(\gamma)$ preserves discrete objects. \square

When \mathbf{C} is tight, the discrete objects may be characterized categorically within $\mathbf{Asm}_t(\mathbf{C})$ (whence the functors $\mathbf{Asm}_t(\gamma)$ arising from discrete simulations may also be characterized categorically). By a *singleton assembly* we mean any assembly U such that $\Gamma(U)$ is a singleton set.

Proposition 5.11 Suppose \mathbf{C} is tight. An assembly X over \mathbf{C} is discrete iff for every singleton object U and set S , every morphism $U \times S \rightarrow X$ in $\mathbf{Asm}_t(\mathbf{C})$ is constant.

PROOF: The forwards implication is straightforward. For the converse, if X is not discrete, take $x \neq x'$ and a such that $a \Vdash_X x$ and $a \Vdash_X x'$. Let U be the singleton assembly with just the realizer a , let S be a two-element set, and let $f : U \times S \rightarrow X$ map the two elements of $|U \times S|$ to x and x' respectively. Since \mathbf{C} is tight, f is tightly tracked by the appropriate identity relation, and so is a non-constant morphism of $\mathbf{Asm}_t(\mathbf{C})$. \square

There does not appear to be a correspondingly simple characterization of the discrete objects within $\mathbf{Asm}(\mathbf{C})$. It is also worth noting that the tight setting preserves a familiar feature from the classical theory of realizability: namely, that a morphism into a discrete assembly is uniquely determined by any tracker for it.

Two further facts about discrete simulations in the tight setting is worth recording. The proofs are straightforward.

Proposition 5.12 (i) If $\gamma : \mathbf{C} \multimap \mathbf{D}$ and $\delta : \mathbf{D} \multimap \mathbf{C}$ are tight simulations with $\delta \circ \gamma \preceq \text{id}_{\mathbf{C}}$, then γ is discrete.

(ii) If $\gamma : \mathbf{C} \multimap \mathbf{D}$ is discrete and \mathbf{D} is deterministic, then \mathbf{C} is deterministic. \square

Combining the two halves of this yields a useful strengthening of Proposition 5.7.

Next, we consider a property of simulations we shall call *univalence*. Although the name is perhaps not ideal, the intuition is that these are simulations that are ‘essentially single-valued’. This notion comes in a non-tight and a tight flavour. The notion of a single-valued simulation was defined in Definition 4.3; similarly, we call an assembly X single-valued if for each $x \in |X|$ there is a unique a with $a \Vdash_X x$.

Definition 5.13 (i) A simulation $\gamma : C \multimap D$ is univalent [resp. tightly univalent] if for some single-valued simulation γ' we have $\gamma \preceq \gamma'$ [resp. $\gamma \preceq_t \gamma'$].

(ii) An assembly X over C is univalent [resp. tightly univalent] if for some single-valued assembly X' we have $X \cong X'$ in $\mathbf{Asm}(C)$ [resp. in $\mathbf{Asm}_t(C)$].

Proposition 5.14 (i) $\gamma : C \multimap D$ is univalent iff $\mathbf{Asm}(\gamma)$ preserves univalent assemblies.

(ii) γ is tightly univalent iff $\mathbf{Asm}_t(\gamma)$ preserves tightly univalent assemblies.

PROOF: (i) The forwards implication is easy. For the reverse, suppose $\gamma_* = \mathbf{Asm}(\gamma)$ preserves univalent assemblies. For each $A \in |C|$, let $Z_A \in \mathbf{Asm}(C)$ be the corresponding object of realizers, and choose $Y_A \in \mathbf{Asm}(D)$ single-valued such that $|Y_A| = A$ and $Y_A \cong \gamma_*(Z_A)$ via the identity on underlying sets. Define $\gamma' : C \multimap D$ by $\gamma'_A(a, b) \Leftrightarrow b \Vdash_{Y_A} a$. We need to show that γ' is a simulation; it is then clear that $\gamma' \preceq \gamma$.

Given $r \in C[A, B]$, construct the following diagram in $\mathbf{Asm}(C)$ as in the proof of Proposition 3.17:

$$\begin{array}{ccc} Z_R & \xrightarrow{\tilde{t}} & Z_S \\ \downarrow & & \downarrow \\ Z_A \times B & & Z_B \times A \end{array}$$

Apply γ_* to obtain a diagram Δ in $\mathbf{Asm}(D)$. Now construct a diagram Δ' in $\mathbf{Asm}(D)$ isomorphic to Δ as follows: replace $\gamma_*(Z_A), \gamma_*(Z_B)$ by Y_A, Y_B respectively, then replace the copies and subobjects in turn by the corresponding canonical ones in $\mathbf{Asm}(D)$. Let f be the morphism in Δ' corresponding to $\gamma_*(\tilde{t})$ in Δ , and take r' tracking f . It is easy to see that r' tracks r with respect to γ' .

The proof of (ii) is similar. \square

In contrast to the situation for discrete assemblies, it is the non-tight setting that seems more friendly to univalent assemblies. This is illustrated by the following categorical characterization of univalent objects within $\mathbf{Asm}(C)$ (the proof involves the Axiom of Choice):

Proposition 5.15 An assembly X over C is univalent iff (in $\mathbf{Asm}(C)$) for every morphism $f : X \rightarrow Z$ and every quotient morphism $h : Y \twoheadrightarrow X$ there exists $\bar{f} : X \rightarrow Y$ such that $h \circ \bar{f} = f$.

PROOF: First suppose X is univalent; we may as well assume it is single-valued. Suppose f and h are as above with f tracked by r . We define \bar{f} as follows: For each $x \in |X|$, choose some $a \in A_X$ and $b \in A_Z$ such that $a \Vdash_X x, b \Vdash_Z f(x)$ and $r(a, b)$; then choose $y \in |Y|$ such that $h(y) = f(x)$ and $b \Vdash_Y y$, and set $\bar{f}(x) = y$. Clearly \bar{f} is tracked by r and $h \circ \bar{f} = f$.

For the converse, suppose X has the given quotient property, and consider the assembly Y where $|Y|$ is the relation \Vdash_X considered as a set of pairs (a, x) , and $a' \Vdash_Y (a, x)$ iff $a' = a$. Let $h : Y \twoheadrightarrow X$ be the quotient map given by second projection (this is tracked by a superidentity), and let $f : X \rightarrow X$ be the identity morphism. Take $\bar{f} : X \rightarrow Y$ such that $h \circ \bar{f} = f$. Consider the image of \bar{f} as a subset of $|Y|$, and lift this to a subobject $s : X' \twoheadrightarrow Y$; then \bar{f} factors through s and we have an isomorphism $X \cong X'$ with X' single-valued. \square

Remark 5.16 In the PCA setting, the quotient maps are precisely the regular epis, so the quotient property featuring in the above proposition is precisely the property of being a (*regular*) *projective*. For this reason, univalent simulations and assemblies were referred to as *projective* in [12]. However, we hesitate to adopt this terminology in our present setting, not only because not all quotients need be regular epis in general, but also because there does not appear to be an analogue of the above proposition for $\mathbf{Asm}_t(\mathbf{C})$.

The following useful fact generalizes Theorem 2.5.3(ii) of [12]:

Theorem 5.17 *Suppose \mathbf{D} is deterministic. If $\gamma : \mathbf{C} \multimap \mathbf{D}$ and $\delta : \mathbf{D} \multimap \mathbf{C}$ are simulations with $\delta \circ \gamma \preceq \text{id}_{\mathbf{C}}$ and $\text{id}_{\mathbf{D}} \preceq \gamma \circ \delta$, then δ is univalent.*

PROOF: For each $B \in |\mathbf{D}|$, take $u_B \in \mathbf{D}[B, \gamma\delta B]$ tracking $\text{id}_{\mathbf{D}} \preceq \gamma \circ \delta$ at B . Since \mathbf{D} is deterministic, for each $b \in B$ there is a unique d_b with $u(b, d_b)$, and it satisfies $(\gamma\delta)_B(b, d_b)$. Moreover, there is a unique $c_b \in \delta_B$ with $\delta_B(b, c_b)$ and $\gamma_{\delta B}(c_b, d_b)$, since γ is discrete by Proposition 5.12. Define $\delta' : \mathbf{D} \multimap \mathbf{C}$ by $\delta'_B(b, c) \Leftrightarrow c = c_b$. We shall show that δ, δ' are ‘intertransformable’ as relations, and hence that δ' is a simulation.

Clearly $\delta'_B(b, c)$ implies $\delta_B(b, c)$. Conversely, if $\delta_B(b, c)$, take $u'_B \in \mathbf{C}[\delta B, \delta\gamma\delta B]$ tracking u_B with respect to δ ; then for some $c' \in \delta\gamma\delta B$ we have $u'_B(c, c')$ and $\delta_{\gamma\delta B}(d_b, c')$, whence $(\delta\gamma)_{\delta B}(c_b, c')$. Now take t_B tracking $\delta \circ \gamma \preceq \text{id}_{\mathbf{C}}$ at δ_B ; then $t_B(c', c_b)$. So taking $v_B \in \mathbf{C}[\delta B, \delta' B]$ a supercomposite of u'_B and t_B , we have that $v_B(c, c_b)$. It is now easy to see that δ' is a simulation: for any $r \in \mathbf{D}[A, B]$, take $r' \in \mathbf{C}[\delta A, \delta B]$ tracking r with respect to δ , and take $r'' \in \mathbf{C}[\delta' A, \delta' B]$ a supercomposite of r' and v_B (noting that $\delta' A = \delta A$); then clearly r'' tracks r with respect to δ' . Moreover, $\delta' \preceq \delta$ is tracked at B by a superidentity, and $\delta \preceq \delta'$ is tracked at B by v_B . \square

Corollary 5.18 *If $\mathbf{C} \simeq \mathbf{D}$ in $\mathcal{CSTRUCT}_t$, then both halves of the equivalence are discrete and univalent; hence there is an equivalence $\mathbf{C} \simeq \mathbf{D}$ consisting of single-valued simulations.*
 \square

In the light of Remark 3.5(ii), this shows that for deterministic \mathbf{C} -structures, our notion of equivalence agrees precisely with the *lax simulation equivalence* of [4] (taking sets and partial functions as the base category).

5.4 \mathbf{C} -structures with products

We see it as a matter of some pride that we have not so far required our \mathbf{C} -structures to be equipped with *products* of any kind at all. Nevertheless, most naturally arising \mathbf{C} -structures do have products of some kind, so it is natural to consider various possible notions of product structure. We opt here for a simple definition with a relatively ‘strict’ flavour.

Definition 5.19 *Suppose \mathbf{C} is a \mathbf{C} -structure.*

(i) *We say \mathbf{C} is strict monoidal if*

- $|\mathbf{C}|$ contains the singleton set $I = \{*\}$ and is closed under set-theoretic binary products;
- for any $r \in \mathbf{C}[A, B]$ and $s \in \mathbf{C}[C, D]$, the relation ‘ $r \times s$ ’ is present in $\mathbf{C}[A \times C, B \times D]$;

- the canonical bijections $(A \times B) \times C \cong A \times (B \times C)$, $I \times A \cong A$ and $A \times I \cong A$ are all present as relations in \mathbf{C} in both directions.

(ii) We say \mathbf{C} has strict cartesian if, in addition to the above, the canonical projection and diagonal maps $A \times B \rightarrow A$, $A \times B \rightarrow B$, $A \rightarrow A \times A$ are present as relations in \mathbf{C} .

Other more relaxed notions of monoidal or cartesian \mathbf{C} -structure may be devised, but it is typically the case that any such \mathbf{C} -structure will be equivalent to a strict one (we refrain from going into details here). For example, all TPCAs (including untyped PCAs) are equivalent to strict cartesian \mathbf{C} -structures, although they may not be such structures themselves.

Clearly, if \mathbf{C} is strict monoidal [resp. strict cartesian] then $\mathbf{Asm}(\mathbf{C})$ is a monoidal [resp. cartesian] category (as is $\mathbf{Asm}_t(\mathbf{C})$ in the case that \mathbf{C} is tight).

The relevant structure-respecting simulations for strict monoidal \mathbf{C} -structures are the following:

Definition 5.20 Suppose \mathbf{C}, \mathbf{D} are strict monoidal. A simulation $\gamma : \mathbf{C} \multimap \mathbf{D}$ is monoidal if

- for any $A, B \in |\mathbf{C}|$ there is a relation $t \in \mathbf{D}[\gamma A \times \gamma B, \gamma(A \times B)]$ such that

$$\gamma_A(a, a') \wedge \gamma_B(b, b') \Rightarrow \exists c'. t((a', b'), c') \wedge \gamma_{A \times B}((a, b), c')$$

- there exists $u \in \mathbf{D}[I, \gamma I]$ such that $u(*, a)$ for some $a \Vdash_I *$.

Recall that if $(\mathcal{C}, \otimes, I)$ and $(\mathcal{D}, \otimes', I')$ are monoidal categories, a lax monoidal functor between them is a functor $F : \mathcal{C} \rightarrow \mathcal{D}$ together with a family of morphisms $\phi_{A,B} : F(A) \otimes' F(B) \rightarrow F(A \otimes B)$, natural in A and B , and a morphism $\psi : I' \rightarrow F(I)$, such that certain coherence conditions involving the associativity and unit morphisms are satisfied. We record the following without proof:

Proposition 5.21 Suppose \mathbf{C}, \mathbf{D} have strict finite monoidal products. A simulation $\gamma : \mathbf{C} \multimap \mathbf{D}$ is monoidal iff $\mathbf{Asm}(\gamma)$ is lax monoidal.

Note also that if the products in \mathbf{C}, \mathbf{D} are cartesian, then a lax monoidal functor $\mathbf{Asm}(\gamma)$ is automatically cartesian (i.e. preserves finite products).

We may write $\mathcal{CSTRUCT}_m$ for the preorder-enriched category of strict monoidal \mathbf{C} -structures and monoidal simulations. We say \mathbf{C}, \mathbf{D} are monoidally equivalent ($\mathbf{C} \simeq_m \mathbf{D}$) if they are equivalent in $\mathcal{CSTRUCT}_m$ (it does not seem that $\mathbf{C} \simeq \mathbf{D}$ implies $\mathbf{C} \simeq_m \mathbf{D}$). The 2-category $\mathcal{CSTRUCT}_m$ has finite products as in $\mathcal{CSTRUCT}$, and it is routine to verify that a monoidal analogue of the construction of near-exponentials goes through. Note, however, that our simple construction of finite sums is not available in $\mathcal{CSTRUCT}_m$, since $\mathbf{C} + \mathbf{D}$ will not typically contain products $A \times B$ where $A \in |\mathbf{C}|$, $B \in |\mathbf{D}|$.

Finally, we remark that analogous results hold in the ‘tight’ setting. If \mathbf{C} and \mathbf{D} are tight and strict monoidal, we may say a tight simulation $\gamma : \mathbf{C} \multimap \mathbf{D}$ is tightly monoidal if it satisfies the obvious tightened version of Definition 5.20. We then have that γ is tightly monoidal iff $\mathbf{Asm}_t(\gamma)$ is lax monoidal; such a functor is automatically cartesian if \mathbf{C} and \mathbf{D} are. The corresponding 2-category $\mathcal{CSTRUCT}_{mt}$ enjoys properties similar to $\mathcal{CSTRUCT}_m$.

5.5 Higher order C-structures

With the machinery of products now in place, we may now identify a class of C-structures that model *higher order* notions of computation, and thus retrieve the theory of TPCAs as a special case of our theory. The key idea here is that a C-structure is higher order if its computable operations can themselves be represented as values of an appropriate type.

Definition 5.22 *A higher order C-structure is a monoidal C-structure \mathbf{C} equipped with the following additional structure: for each $A, B \in |\mathbf{C}|$, a set $(A \Rightarrow B) \in |\mathbf{C}|$ and a relation $app_{AB} \in \mathbf{C}[(A \Rightarrow B) \times A, B]$ such that*

$$\text{for any } r \in \mathbf{C}[C \times A, B] \text{ there exists a total relation } \bar{r} \in \mathbf{C}[C, (A \Rightarrow B)] \text{ such that} \\ r \subseteq (\bar{r} \times \text{id}_A); app_{AB}.$$

It is crucial here that \bar{r} is not required to be unique; we may therefore say app_{AB} enjoys a ‘weak universal property’.

It is not hard to show that if \mathbf{C} is a C-structure [resp. cartesian C-structure] then $\mathbf{Asm}(\mathbf{C})$ is monoidal closed [resp. cartesian closed]. Thus, as is often the case in realizability, the \mathbf{Asm} construction turns a weak universal structure into a strong one.

Any TPCA may be turned into a deterministic, higher order cartesian C-structure $\mathbf{CStr}(C)$ as follows. First, it is easy to construct some C' equivalent to C (within $TPCA$) in which product types are genuine set-theoretic products, and in which a unit type exists. Such a C' may then be viewed as a C-structure $\mathbf{CStr}(C)$ as in Example 3.3(i). Clearly $\mathbf{CStr}(C)$ is deterministic and cartesian, and for any types A, B , the element

$$\lambda^*p.(fst\ p)(snd\ p) \in (((A \Rightarrow B) \times A) \Rightarrow B)$$

induces a computable operation $app_{AB} : (A \Rightarrow B) \times A \rightarrow B$ with the required weak universal property. (We assume familiarity here with the λ^* notation for combinators — see e.g. [12, Chapter 1].)

In one important respect, however, deterministic higher order cartesian C-structures are significantly more general than TPCAs. In C-structures of the form $\mathbf{CStr}(C)$, every element a of any $A \in |\mathbf{C}|$ is picked out by some operation in $\mathbf{C}[I, A]$ (We may call such elements *a computable*, since the idea is that mappings in $\mathbf{C}[I, A]$ are ‘computable operations’.) In general, however, this need not be the case — rather the elements picked out by such operations will constitute a sub-C-structure of \mathbf{C} in an evident sense. This amounts to the observation that deterministic higher order cartesian C-structures naturally embrace the concept of *relative realizability* (see e.g. [2]). Informally, the idea behind relative realizability is to consider (for instance) a TPCA A equipped with a sub-TPCA A^\sharp of elements that we regard as ‘computable’. Under mild restrictions,³ it can be shown that deterministic higher order cartesian C-structures correspond exactly to TPCAs equipped with such a choice of sub-TPCA.

In any case, this convergence between TPCAs and certain C-structures suggests how the notion of applicative morphism for TPCAs (as in Section 2.2) might be generalized to arbitrary higher order C-structures:

³The mild restrictions needed to get a perfect correspondence are that we should consider TPCAs satisfying the stronger condition $s \cdot x \cdot y \cdot z \simeq x \cdot z \cdot (y \cdot z)$, and higher order C-structures in which the operation \bar{r} satisfies $r = (\bar{r} \times \text{id}_A); app_{AB}$.

Definition 5.23 Suppose \mathbf{C}, \mathbf{D} are higher order \mathcal{C} -structures. An applicative morphism $\gamma : \mathbf{C} \rightarrow \mathbf{D}$ consists of a mapping $A \mapsto \gamma A : |\mathbf{C}| \rightarrow |\mathbf{D}|$ and a family of total relations $\gamma_A \subseteq A \times \gamma A$ (for $A \in |\mathbf{C}|$) satisfying the following conditions:

1. For every computable element $a \in A$ (where $A \in |\mathbf{C}|$) there is some computable $a' \in \gamma A$ such that $\gamma(a, a')$.
2. ‘Application in \mathbf{C} is tracked in \mathbf{D} ’: that is, for any $A, B \in |\mathbf{C}|$, we may choose $app_{AB} \in \mathbf{C}[(A \Rightarrow B) \times A, B]$ as in Definition 5.22 such that for some $app'_{AB} \in \mathbf{D}[\gamma(A \Rightarrow B) \times \gamma A, \gamma B]$ we have

$$\gamma_{(A \Rightarrow B)}(f, f') \wedge \gamma_A(a, a') \wedge app_{AB}((f, a), b) \Rightarrow \exists b'. \gamma_B(b, b') \wedge app'_{AB}((f', a'), b')$$

If \mathbf{C}, \mathbf{D} are TPCAs, it is easy to see that applicative morphisms $\mathbf{CStr}(\mathbf{C}) \rightarrow \mathbf{CStr}(\mathbf{D})$ in the above sense correspond exactly to applicative morphisms $\mathbf{C} \multimap \mathbf{D}$ in the sense of Section 2.2. Indeed, if $r \in D_{\gamma(A \Rightarrow B) \Rightarrow \gamma A \Rightarrow \gamma B}$ tracks \cdot_{AB} in the TPCA sense, then $\lambda^* p. r(fst p)(snd p)$ induces a suitable operation $app'_{AB} \in \mathbf{CStr}(\mathbf{D})[(A \Rightarrow B) \times A, B]$; conversely, any such operation app'_{AB} is induced by some $r' \in D_{\gamma(A \Rightarrow B) \times \gamma A \Rightarrow \gamma B}$, and then $\lambda^* f a. r'(pair f a)$ tracks \cdot_{AB} in the TPCA sense. So Definition 5.23 is a natural and direct generalization of the TPCA notion of applicative morphism.

The connection with simulations is now given by the following:

Proposition 5.24 If \mathbf{C}, \mathbf{D} are higher order \mathcal{C} -structures, then the applicative morphisms from \mathbf{C} to \mathbf{D} are exactly the monoidal simulations $\mathbf{C} \multimap \mathbf{D}$.

PROOF: First suppose γ is a monoidal simulation. To see that condition 1 of Definition 5.23 is satisfied, note that if $a \in A$ is picked out by $r \in \mathbf{C}[I, A]$ and r is tracked by $r' \in \mathbf{D}[\gamma I, \gamma A]$, then by supercomposing r with the operation $u \in \mathbf{D}[I, \gamma I]$ from Definition 5.20 yields an operation in $\mathbf{D}[I, \gamma A]$ that picks out a γ -realizer for a . For condition 2, given any $A, B \in |\mathbf{C}|$ we have an operation $t \in \mathbf{D}[\gamma(A \Rightarrow B) \times \gamma A, \gamma((A \Rightarrow B) \times A)]$ as in Definition 5.20, and an operation $u \in \mathbf{D}[\gamma((A \Rightarrow B) \times A), \gamma B]$ tracking the operation app_{AB} in \mathbf{C} . Supercomposing these gives an operation $app'_{AB} \in \mathbf{D}[\gamma(A \Rightarrow B) \times \gamma A, \gamma B]$ with the required properties.

Conversely, suppose γ is an applicative morphism. Given any $r \in \mathbf{C}[A, B]$, supercompose with the bijection in $\mathbf{C}[I \times A, A]$ to obtain $r' \in \mathbf{C}[I \times A, B]$, then choose $\bar{r} \in \mathbf{C}[I, (A \Rightarrow B)]$ as in Definition 5.22. Let $f \in (A \Rightarrow B)$ be any computable element picked out by \bar{r} ; then by condition 1 of Definition 5.23, we may find $f' \in \gamma(A \Rightarrow B)$ picked out by some $t \in \mathbf{D}[I, \gamma(A \Rightarrow B)]$ such that $\gamma_{(A \Rightarrow B)}(f, f')$. By supercomposing the canonical operation in $\mathbf{D}[A, I \times A]$ with $t \times i \in \mathbf{D}[I \times A, (A \Rightarrow B) \times A]$ and then with app'_{AB} , we obtain an operation in $\mathbf{D}[A, B]$ that tracks r . \square

The basic idea here — that if the application operations are tracked then so are all operations — is one that also plays an important role in [3].

Under the above correspondence, it is easy to see that the preorder \preceq on monoidal simulations agrees precisely with the one on applicative morphisms as in Section 2.2. Thus, the \mathbf{CStr} construction outlined above extends to a 2-functor $\mathbf{CStr} : \mathbf{TPCA} \rightarrow \mathbf{CSTRUCT}_m$ which is locally an equivalence. (Hence two TPCAs A, B are equivalent iff

$\mathbf{CStr}(A)$, $\mathbf{CStr}(B)$ are equivalent as monoidal \mathbf{C} -structures.) Moreover, as noted earlier, the \mathbf{Asm} construction on TPCAs agrees with that on \mathbf{C} -structures modulo this inclusion.

The observation that the notion of applicative morphism is in essence a ‘first order’ one highlights a significant philosophical point. One might have supposed that two models of higher order computation could be equivalent as ‘flattened’ first order models but differ non-trivially on account of the different higher order structures they carried. Proposition 5.24 shows that this is not the case, at least within our present framework: equivalence of higher order models is simply their equivalence *qua* first order models. Thus, the task of classifying notions of higher order computability (as advocated e.g. in [15, 16]) can be seen just as part of the larger task of classifying computability notions in general.

Note, however, that the class of higher order \mathbf{C} -structures appears to have poor closure properties by comparison with that of all \mathbf{C} -structures. Not only are higher order \mathbf{C} -structures not closed under finite sums, but they do not appear to admit an analogue of our near-exponential construction. Curiously, then, it is the extension from higher order to first order structures that allows ‘higher order’ phenomena to appear at the framework level.

6 Conclusions and further work

In summary, we have presented a broad framework for the study of models of computation and simulations between them, and shown that the ‘category of assemblies’ construction and associated correspondence theorem can be generalized to this setting. We have also seen that our category of models possesses some non-trivial mathematical structure, as well as a number of well-behaved subcategories reflecting interesting properties of particular models and simulations. Whilst our investigation has been far from exhaustive, we believe our results sufficiently demonstrate the mathematical richness and fecundity of our proposed framework.

We have also briefly indicated how a wide range of computation models (such as process calculi) naturally fall within our framework. Indeed, we are hopeful that its scope will turn out to be considerably wider, embracing more recent paradigms such as quantum computation and membrane computation. Our hope, then, is that our framework might play some role in allowing us to grasp the ‘big picture’ of how all these approaches to modelling computation are related, and how different ‘levels of description’ of the same computational system can be fitted together. By offering a uniform language within which a multitude of results from diverse areas may be expressed, one can imagine our framework playing an organizing role somewhat akin to that of category theory itself.

Going somewhat further, we are optimistic that by enabling us to consider models up to simulation equivalence, our framework might lead to the identification of natural ‘computability notions’ underlying existing models of computation. For instance, if several superficially quite different process calculi turned out to be equivalent in our setting, this might lead us to suspect that the underlying notion of computability was somehow a fundamental one. We thus have the possibility that the author’s research programme of identifying and classifying natural notions of higher order computability could in principle be extended to computability notions of a much more general kind.

Indeed, the author feels that even as regards the classification of natural higher order

notions is concerned, the present work sheds new light on the endeavour. We have seen that among all C-structures, the class of higher order C-structures does not enjoy particularly good closure properties, suggesting that in some ways this is a less natural and well-rounded class than the class of all C-structures. Combining this with the observation that equivalence of higher order models is nothing more than equivalence as first order models, the moral would seem to be that the task of classifying higher order notions is a less ‘natural’ one than it appeared to us at first: it simply amounts to classifying those first order notions that happen to enjoy a certain somewhat *ad hoc* property. Of course, one can still justify the endeavour on the grounds that *some* limiting criterion is initially desirable if we are to make progress — we have no handle whatever on the range of natural first order notions at present — but we now think that as an ultimate objective, it is the mapping out of computability notions in general that constitutes the more natural goal.

6.1 Some possible alternatives

Naturally, in the context of a research programme such as the one we have outlined, a critical attitude towards our choice of framework is appropriate. Is our 2-category *CSTRUCT* (or any of the variants we have mentioned) the ‘right’ setting for such an undertaking? We mention here two different questions that might be raised regarding possible alternative choices.

Firstly, although we stated at the outset that we were in some sense seeking the ‘most general’ setting in which the **Asm** construction and correspondence theorem go through, we have already noted that a yet more general framework seems to be possible, namely one based on *ordered C-structures*, an easy adaptation to our setting of the notion of ordered PCA (see e.g. [6]). We have chosen not to avail ourselves of this possibility here, partly because we also have an interest in finding the *simplest* setting for our theory (the ordered setting seems to us to involve a modest technical overhead), and also because we are not aware of any models of *computational* interest that naturally constitute ordered C-structures rather than standard ones. Nevertheless, the possibility of broadening our scope to ordered C-structures if necessary should be borne in mind.

Secondly, and perhaps more seriously, there is a significant choice involved in what the notion of ‘simulation’ should be — in the terminology of [4], the choice between *lax* and *strict* simulations. Our present Definition 3.4 corresponds to the lax notion: if r' tracks r , we do not require that the degree of partiality of r' exactly matches that of r . To obtain the strict notion of simulation, we would need to add a condition such as

$$\gamma_A(a, a') \wedge r'(a', b') \Rightarrow \exists b. r(a, b) \wedge \gamma_B(b, b')$$

(This is not to be confused with the notion of tight simulation as in Definition 5.1 — for example, all single-valued simulations are tight but not all are strict.) It is known that equivalence of C-structures under strict simulations is a stronger condition than equivalence under lax simulations (see [3, Section 5]). Whereas lax equivalence coincides with equivalence of the associated *categories of assemblies*, one of the main results of [3] is that strict equivalence coincides with equivalence of *Turing categories* (modulo Morita equivalence).

In [3, 4], Cockett and Hofstra compellingly argue that strict equivalence is more appropriate where questions of *computability* are concerned, while the lax notion is better adapted

to questions of *realizability*. This is because in realizability it is never a problem if a realizer does ‘more’ than the job it is designed for, whereas in computability one is often interested in the precise domain of a computable operation. Certainly, if the idea is that equivalent structures should share ‘the same’ computability theory in some sense, then there will be significant aspects of our structures (such as their theory of m-reducibility) that are stable under strict equivalences but not lax ones. So if we consider models modulo lax equivalence, we are at best dealing with ‘computability notions’ in a somewhat coarse-grained sense.

The main reason we have adopted the lax point of view is that at the level of generality of C-structures, this seems to us (at present) more mathematically fruitful than the strict one. Not only has it been our stated aim to generalize the **Asm** construction and its associated theory to the realm of C-structures, but it would seem that there is rather little of interest to say by way of generalization of Turing categories to this setting: most of the interest in Turing categories (as presented in [3]) seems essentially tied to their higher order nature. We also believe that our more coarse-grained view is probably adequate in practice for arriving at an overall map of the computability landscape; indeed, we do not know whether computationally ‘natural’ examples of models that are laxly not strictly equivalent arise with any great frequency.

6.2 Further work

It is likely that there is further categorical structure to be uncovered within *CSTRUCT*. Certainly there are other subcategories of interest that we have not touched on here: for example, it is natural to look at C-structures that contain a pair of elements playing the role of the booleans. The relevant boolean-respecting simulations between such structures correspond to what are called *decidable* morphisms in [12].⁴ The subcategory of such simulations would itself appear to have reasonable structure and closure properties. Likewise, one could consider C-structures that contain a representation of the natural numbers, and simulations that respect these (these play an important role in [16]).

In passing, we mention that there are two superficially natural conditions on C-structures that are known *not* to correspond to essential properties of the underlying computability notion, in that they are not stable under (lax) equivalences. Let us call a C-structure *C* *total* if every relation $r \in C[A, B]$ is total. And in the higher order setting, let us call *C* *extensional* if no two elements of $(A \Rightarrow B)$ induce the same operation in $C[A, B]$. It is shown in [22] that every total PCA is equivalent to a non-total one (in \mathcal{PCA} , and hence in *CSTRUCT*), and likewise every extensional PCA is equivalent to a non-extensional one.

Aside from such extensions to our theoretical development, however, the main task is to populate our abstract framework with some interesting examples (outside the domain of higher order C-structures, which has already received significant attention). Whilst it seems clear enough that very many kinds of computation models will give rise to C-structures, it would be interesting to find examples of models that are non-trivially equivalent, or of models that are inequivalent for some computationally illuminating reason. It seems to us that the realm of *process calculi* would provide a suitable domain in which to start such an investigation: one could begin by examining a range of known process calculi and

⁴It is remarked in [22] that the name ‘decidable’ was not very well chosen — a point we are happy to concede. Perhaps ‘boolean-respecting’ would be more appropriate.

the possible ways in which they give rise to C-structures, then investigating what kinds of simulations exist between these, and so forth.

Another possible application area that intrigues us concerns the relationship between computation and physics. There appears to be a growing trend in the theory of computation that emphasizes its physical aspects — that is to say, the way in which some purely mathematical model of computation is realized in physical terms. It appears plausible that the notion of C-structure is broad enough to encompass models of the evolution of physical systems: crudely, for instance, the elements of a C-structure might correspond to ‘physical states’, while computable operations might correspond to transformations induced by actions from outside the system. A physical realization of some model of computation (for example, a physical computer) might then appear as a *simulation* of the mathematical model in a physical one. Furthermore, it seems likely that physical reality itself can be modelled in this way at many different levels of description whose interrelationships are expressed by simulations. Thus, one can envisage our general ‘theory of computability notions’ as merging with some kind of ‘theory of physical realizability’ — a prospect that would seem to hold philosophical as well as mathematical appeal.

Finally, it would be interesting to find a use for our near-exponential construction on C-structures. We have already noted that this construction tends to yield wild and intractable structures; however, it appears that an analogous construction in the subcategory of *boolean-respecting* simulations yields much tamer ones. In this setting, the near-exponential $K_1^{K_1}$ is the one-element C-structure; and if L is the PCA of closed λ -terms modulo β -equality, the near-exponential L^L appears to be a rather interesting structure that we have not investigated in detail. As a very tentative possibility, these observations suggest to us the idea of something broadly akin to a *homotopy theory* for computability notions. In classical homotopy theory, one gets at interesting properties of a topological space X via properties of some space of continuous mappings into it (e.g. the ‘loop space’ consisting of mappings from the circle S_1 into X). Could it be that some deeper insight into the essential differences between computability models (for instance, between K_1 and L) might arise from an examination of various near-exponentials constructed from them? On this highly speculative note, we draw our present investigation to a close.

References

- [1] S. Abramsky, Process realizability. In F.L. Bauer and R. Steinbrüggen (eds.), *Foundations of Secure Computation*, IOS Press, 167–180, (2000).
- [2] L. Birkedal and J. van Oosten, Relative and modified relative realizability. *Annals of Pure and Applied Logic* **118**, 115–132 (2002).
- [3] J.R.B. Cockett and P.J.W. Hofstra, Introduction to Turing categories. *Annals of Pure and Applied Logic* **156(2-3)**, 183–209 (2008).
- [4] J.R.B. Cockett and P.J.W. Hofstra, Categorical simulations. To appear in *Journal of Pure and Applied Algebra*.
- [5] S. Feferman, A language and axioms for explicit mathematics. In *Algebra and Logic*, Lecture Notes in Mathematics **450**, Springer, 87–139 (1975).

- [6] P.J.W. Hofstra, All realizability is relative. *Math. Proc. Camb. Phil. Soc.* **141**, 239–264 (2006).
- [7] N. Hoshino, Linear realizability. In *Computer Science Logic 2007*, Lecture Notes in Mathematics **4646** of LNCS, Springer, 420–434 (2007).
- [8] J.M.E. Hyland, The effective topos. In *L.E.J. Brouwer Centenary Symposium*, North-Holland, 165–216 (1982).
- [9] J.M.E. Hyland, First steps in synthetic domain theory. In Carboni *et al.*, eds., *Category Theory, proceedings, Como 1990*, Lecture Notes in Mathematics **1488**, Springer, 131–156 (1990).
- [10] G. Kreisel, Some reasons for generalizing recursion theory. *Logic Colloquium '69*, Studies in Logic and the Foundations of Mathematics **61**, North-Holland, 139–198 (1971).
- [11] P. Lietz and T. Streicher, Impredicativity entails untypedness. *Math. Struct. Comp. Sci.* **12**, 335–347 (2002).
- [12] J.R. Longley, *Realizability Toposes and Language Semantics*. PhD thesis, University of Edinburgh, 1995.
- [13] J.R. Longley, Matching typed and untyped realizability. In L. Birkedal *et al.* (eds.), Proceedings of Workshop on Realizability, Trento. *Electronic Notes in Theoretical Computer Science* **23(1)**, Elsevier (1999).
- [14] J.R. Longley, Unifying typed and untyped realizability. Unpublished electronic note, available from the author’s web page (1999).
- [15] J.R. Longley, Notions of computability at higher types I. In R. Cori *et al.* (eds.), *Logic Colloquium 2000*, Lecture Notes in Logic **19**, ASL, 32–142 (2005).
- [16] J.R. Longley, On the ubiquity of certain total type structures. *Math. Struct. Comp. Sci.* **17(5)**, 841–953 (2007).
- [17] G. Longo and E. Moggi, Constructive natural deduction and its ‘ ω -set’ interpretation. *Math. Struct. Comp. Sci.* **1**, 215–254 (1991).
- [18] R. Milner, Functions as processes. *Math. Struct. Comp. Sci.* **2**, 119–141 (1992).
- [19] M. Schönfinkel, Über die Bausteine der Mathematischen Logik. *Mathematische Annalen* **92**, 305–316 (1924). Translated in J. van Heijenoort, *A Source Book in Mathematical Logic, 1879–1931*, Harvard University Press, 344–366 (1967).
- [20] P. Taylor, The fixed point property in synthetic domain theory. In *Proc. 6th Annual Symposium on Logic in Computer Science*, IEEE, 152–160 (1991).
- [21] J. van Oosten, Extensional realizability. *Annals of Pure and Applied Logic* **84**, 317–349 (1997).
- [22] J. van Oosten, *Realizability: An Introduction to its Categorical Side*. Studies in Logic and the Foundations of Mathematics **152**, Elsevier, 2008.